State of Michigan

Systems Development Life Cycle (SDLC)

OFFICE OF RESEARCH AND POLICY

NOVEMBER 2001

Version 1.0

Systems Development Lifecycle

The Systems Development LifeCycle (SDLC) integrates software development, project management, and quality assurance processes into a software lifecycle that is controllable, predictable, and repeatable. The lifecycle processes are compatible with SOM policy on software development and maintenance, and compliant with the Capability Maturity Model (CMM) Level 2 and level 3 key process areas in the Software Engineering Institute's model. The lifecycle processes are divided into phases, activities, and tasks that can be combined or modified as necessary to fit the needs of various types and sizes of projects.

The State of Michigan has made a commitment to CMM Level by 2003. With that goal in mind, the purpose of this document is to describe and define the lifecycle approved for use by the State of Michigan.

This document presents guidelines for an organized, disciplined approach to systems development projects that is based on industry standards and best practices. It describes the methods and practices for each phase of the systems development lifecycle that starts with project initiation and ends with project closeout. Additional methods are provided for emergency maintenance. For each defined lifecycle phase, this document presents guidelines for the development process and its management, and for the products produced and their reviews.

Your comments, suggestions, and questions are welcome. We would like to know what works, or does not work, for your project and why. We need your feedback to help identify information that should be included or deleted in future SDLC releases. Please contact:

Department of Information Technology, Office of Research and Policy, Penny Dewey at 517-241-2926
deweyp1@michigan.gov

Points of Contact

Please forward any comments or questions to the Office of Research and Policy, Management within the Department of Information Technology. The Office of Research and Policy can be reached at (517)-373-2635, or visit our Web site at: http://www.michigan.gov/dit

All documents are located in the Enterprise IT Standards folder. The SDLC Framework folder contains SDLC phases, checklists and templates.

Acknowledgements

The State of Michigan would like to acknowledge the following individuals and organizations that created this Systems Development Lifecycle. Without their input and hard work, this would not have been achieved.

INDIVIDUALS

Penny Dewey DIT – Research and Policy	**David Hill** DMB, Office of Information Technology Services Division
Larry Bailey Department of Environmental Quality	**Sheila Kelley** Department of State
Vaughn Bennett DMB, Office of Project Management	**Christine Knapp** MAIN
Dan Buonodono DMB, Office of Project Management	**Linda Myers** Department of Community Health
Juan Chapa Department of Treasury	**Cathy Pelham** DMB, Office of Information Technology Services Division
Susan Felkowski MAIN	**Lucy Pline** MAIN
John Forslin Department of Natural Resources	**Ginger Schoettinger** Department of State

For ease of handling, these materials are divided into 12 sections and several appendixes. Guidelines have been provided to allow tailoring of your project based on size and complexity.

Sections 1-11 contain information on the lifecycle. Section 12 contains information about Emergency Maintenance. Checklists and templates are provided to assist in the preparation of the deliverables. Some reference has been made to the Project Management Methodology (PMM) where appropriate.

Pink – refers to sections in the SDLC
Green – refers to forms
Red – deliverables of the phase
Blue is optional deliverables or web site

State of Michigan
Systems Development Lifecycle

Systems Development Lifecycle (SDLC)

Service Request

Initiate Project

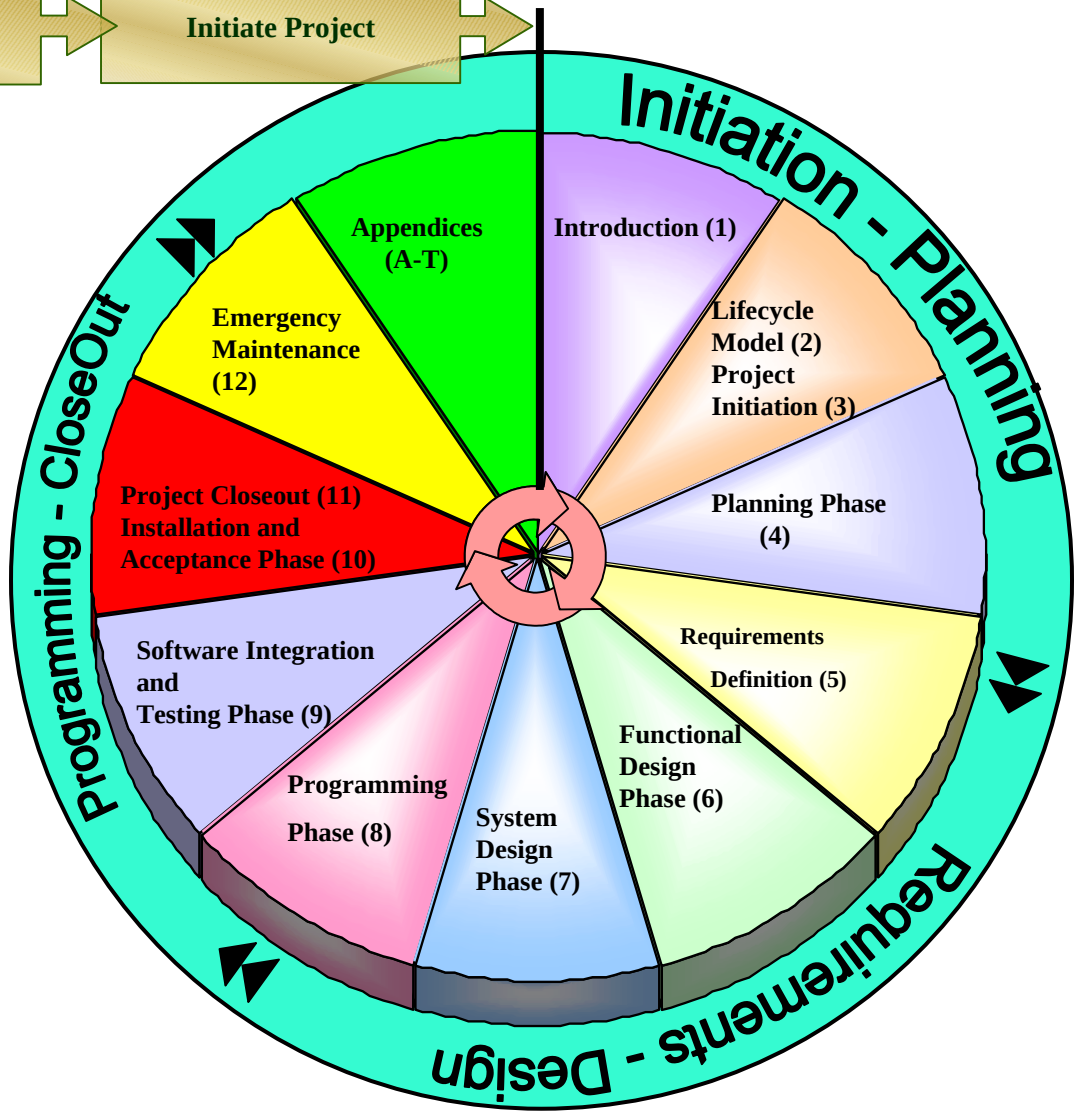


Table of Contents

Section 1 - Introduction

Section 2 – Lifecycle Model

Section 3 – Project Initiation

Section 4 – Planning Phase

Table of Contents

Section 5 – Requirements Definition Phase

Table of Contents

Section 6 – Functional Design Phase

Table of Contents

Section 7 – System Design Phase

Section 8 – Programming Phase

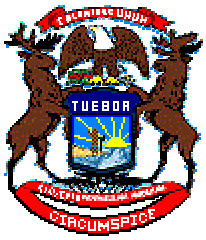
Table of Contents

Section 9 – Software Integration and Testing Phase

Section 10 – Installation and Acceptance Phase

Section 11 – Project Closeout

Table of Contents



SYSTEMS DEVELOPMENT LIFECYCLE

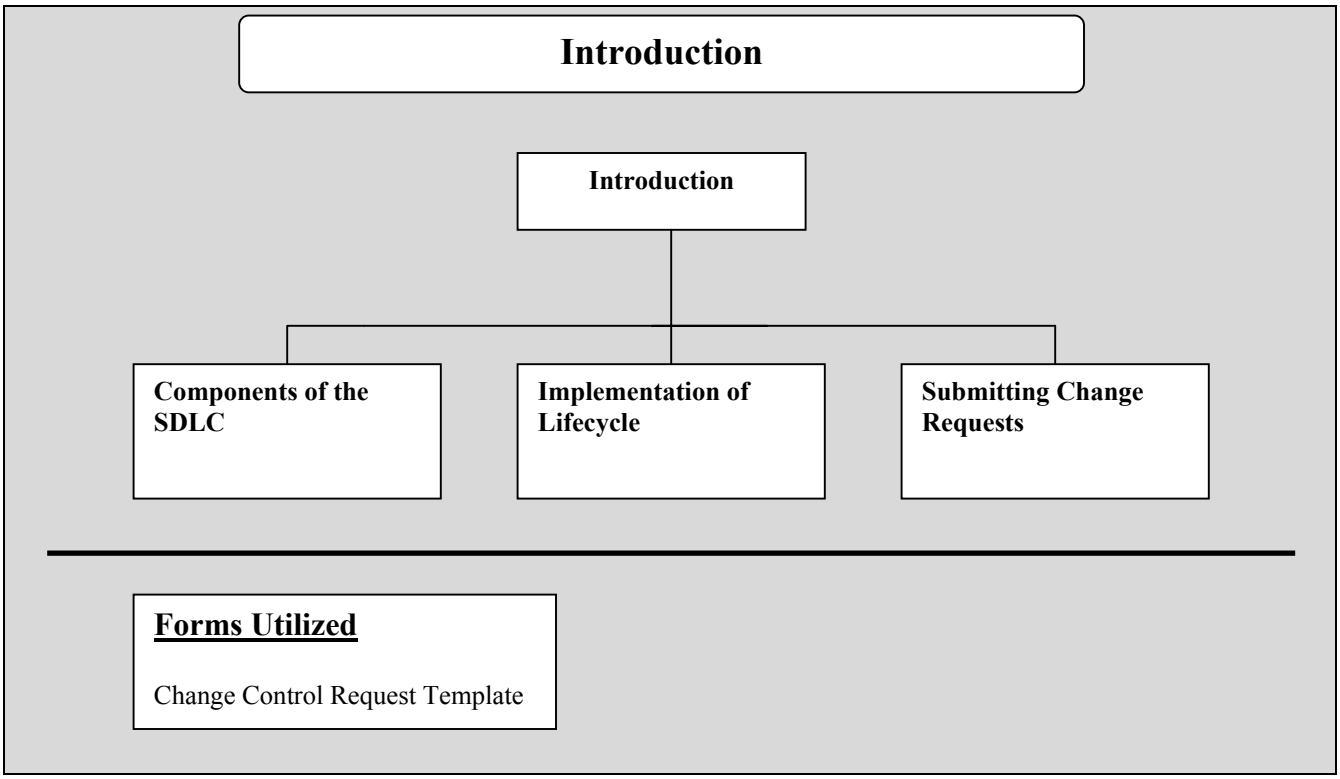
SECTION 1

INTRODUCTION

Table of Contents

Section 1: Introduction

Highlights of Phase



Section 1: Introduction

Overview

Description:

This section presents the standards and principles that form the basis for the State of Michigan's Systems Development Lifecycle (SDLC) processes. The primary purpose of the lifecycle is to promote the development of reliable, cost-effective, computer-based software products while making efficient use of resources. Use of the lifecycle will also aid in the status tracking, management control, and documentation efforts of the project.

This SDLC is consistent with other lifecycles used in the government and private industry. It conforms to the statewide Project Management Methodology (PMM) and the Capability Maturity Model (CMM) components: software configuration management, software quality assurance, software subcontractor management, project tracking and oversight, software project planning, and requirements management. It should be used in compliance with the State's information architecture and initiatives such as e-Michigan and Enterprise Information Technology Standards.

Significant input for the lifecycle was obtained from agencies using enterprise standards development teams. The lifecycle integrates agencies best practices and focuses on the quality of both the software development process and the deliverables generated from the process.

The SDLC is derived from the principles and standards advocated by software quality industry leaders, such as The Institute of Electrical and Electronics Engineers (IEEE) and the Carnegie-Mellon Software Engineering Institute (SEI). This lifecycle is designed to enable project teams to fully achieve Level 2 maturity on the SEI Capability Maturity Model (CMM). Some Level 3 key process areas are also incorporated into the lifecycle.

Software quality assurance is integrated into the SDLC, making quality the responsibility of all project team manager(s) and members. To assure the development of quality software products, the lifecycle prescribes reviews, inspections, and audits for the lifecycle processes and technical deliverables. To protect the integrity of the software, the lifecycle also prescribes configuration controls over software, data, and technical documentation.

The SDLC encompasses all aspects of the systems development lifecycle from project initiation through production and maintenance, and integrates the following basic lifecycle management concepts:

- Implementation of systems development preferred practices using a graded approach based on the level of effort, complexity, and degree of external impact of the software product.
- Implementation of a project management lifecycle including quality assurance, configuration management, and a comprehensive testing approach that is adaptable to the individual development environments.

Description

- Application of a complete documentation approach supporting both

Overview

Continued:

lifecycle and project management activities, to assure an effective method for managing, tracking, and evaluating software development activities.

The SDLC was developed for project managers, project teams, and IT managers who are responsible for developing a new computer-based software product or making enhancements to an existing system. The lifecycle will be reviewed on an annual basis and modified as needed to keep pace with the changing needs of the agencies systems development environment and the continuing technical advances in the information technology industry.

The following sections provide additional information about using this systems development lifecycle:

SDLC Components
Implementation of Lifecycle
Submitting Change Requests

The SDLC can be used on the following types of applications:

- Mainframe
- Client/Server
- Procurement
- Commercial Off-the-Shelf (COTS)
- Emergency Maintenance
- Enhancement
- Web Development

Benefits of Using a Lifecycle:

The benefits of using a clear, well-defined lifecycle increases the probability that:

- The system's performance will meet the customer's needs
- Quality products are developed and maintained
- Resource use is optimized during development
- Systems will comply with relevant standards
- Efficient and effective applications will be built and implemented
- Risk will be accurately assessed and minimized
- The development process will produce a complete set of deliverables per the CMM Level 3 model.

Components of SDLC

Description:

Since the issuance of the first version of the Project Management Methodology (PMM) in May 2000, several State of Michigan agencies have recognized the need for a Systems Development Lifecycle (SDLC) that parallels the PMM to be considered for all software projects. A brief discussion of some of these activities that are applicable to software projects is provided in this section.

Project Management Methodology:

All software projects should be following the guidance provided by the State of Michigan's Project Management Methodology. Specifically, software projects should review information concerning the six sections of the PMM and the appendices, checklists, and templates.

Enterprise Information Technology Standards Program:

The SOM Enterprise Information Technology (IT) Standards Program was established to adopt and implement standards to ensure the wise stewardship of information technology resources and to support the goals of the SOM Information Technology environment. The Enterprise IT Standards Program sponsors an ongoing Information Technology standards adoption and publication process. This information is maintained on the SOM Enterprise IT Standards Web site. (http://www.michigan.gov/dit)

Additionally, the Enterprise IT Standards Program is managed by the Office of Research and Policy.

Enterprise Standards Review Team (ESRT):

The Department of Management and Budget (DMB) is responsible, in consultation with agencies, for ensuring that IT support systems for the state are effective and efficient. Organizationally, this responsibility is delegated to the State CIO. The State CIO meets monthly with agency CIO's as a group called IMPACT (Information Management Policy Advisory Committee) to discuss and formulate statewide information technology policy.

Standardization of IT methods, approaches, equipment, software, and protocols facilitates effective and efficient systems by:

- Enabling coordinated and timely standards development administration
- Ensuring prioritization and support for standards development processes
- Reducing cost of government through efficient and consistent technology management

The purpose of this Enterprise Standards Development Lifecycle Process (SDLP) is to allow IT Standards to evolve in an orderly and repeatable manner while ensuring compatibility and supportability across the state's core IT infrastructure. The SDLP provides a uniform, repeatable mechanism for the development, adoption and application of enterprise IT standards. The process includes a Sponsor to create requests, an Enterprise Standards Review Team (ESRT) to provide guidance and administration of standards development, and Standards Development Teams (SDT's) to review, research, prepare findings, and draft standards for consideration and review by IMPACT.

Components of SDLC

Comparison

The following table depicts the relationship between the systems development lifecycle (SDLC) and project management.

SDLC	Project Management
A framework for *solving technical challenges.* **Focus**: Define the attributes of the desired product. **Who:** What are the technical roles & responsibilities. **Measurement:** Progress against the technical requirements.	A framework for *planning and managing work.* **Focus:** Plan how to deliver product on time / within budget. **Who:** What are the management roles & responsibilities. **Measurement:** Progress against the project plan.

Illustrations in Sections

All illustrations throughout the systems development lifecycle we adopted from the Michigan Administrative Information Network (MAIN) DCDS Time/Activity Reporting System Project.

The objective of this development project is to build an interface that will read the time sheet data from the electronic files received from agencies, who have the time and attendance data maintained in their time clock's time management software or customized time/activity reporting system. Once the data has been received from the state agency, it will then update the DCDS database with the time and/or activity data. This will eliminate the need for time and/or activity data entry currently done in DCDS.

Samples of the objectives, scope, requirements and interfaces will appear throughout the document.

Implementation of Lifecycle

Description:

This lifecycle integrates software development, project management, and quality assurance practices and is designed to be flexible. It can be adapted to accommodate the specific needs of any software development project and all computing platforms used by the agencies including standalone and networked, mainframes, client/server, emergency maintenance, enhancements, web development, procurement and commercial off-the-shelf (COTS) software.

Projects that were initiated prior to the awareness or usage of this document should plan to implement the lifecycle at the earliest feasible time or the next release of the product. If a Project Plan already exists, make the revisions necessary to integrate the systems development lifecycle, project management, and quality assurance practices, as appropriate. If a Project Plan does not exist, develop a plan that summarizes the activities and deliverables of the previous phases and incorporates the lifecycle activities and products into the subsequent phases.

Since the lifecycle does not provide specific guidance for every systems development situation, suggestions for adapting the lifecycle to accommodate projects of varying size, complexity, or criticality are provided in *Section 2 Lifecycle Model*. Samples of project plans for projects using the lifecycle are available on the Web site at: http://www.michigan.gov/dit under *Strategies and Methodologies*.

Questions:

If specific questions are generated concerning the interpretation or applicability of portions of the lifecycle, the project team should attempt to resolve them during the project review activities built into the phases of the lifecycle. The system owner/customer(s) and other project stakeholders must concur with any adaptations that are made.

Office of Research and Policy staff may also be consulted on the interpretation or applicability of the lifecycle.

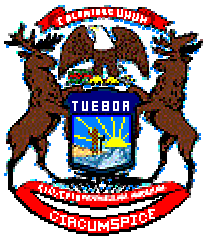
Submitting Change Requests

Description:

The agency systems development environment is continuously changing as emerging technologies are integrated into projects, system owner/customer requirements are expanded, and organizational needs evolve. The systems development lifecycle will be expanded and revised, as needed, to reflect changes in the environment, improvements suggested through customer feedback, and the maturation of software development capabilities.

Customers of the lifecycle are encouraged to submit suggestions for improving its content and to report any practices that are difficult to understand or create an implementation problem for a project team. Suggestions and problems should be submitted on the Change Control Request Template that is provided in the Project Management Methodology. If the form is not available or does not accommodate the type of request being made, submit a memo that describes the suggestion or problem.

The Change Control Request template should be submitted to the Office of Research and Policy. All requests will be evaluated and the originator of the request will be notified of the action taken. Some requests will be handled immediately while others may require investigation by an ad hoc working group of knowledgeable personnel. In some cases, a request may not be appropriate for the current environment, but will be retained for future consideration.



SYSTEMS DEVELOPMENT LIFECYCLE

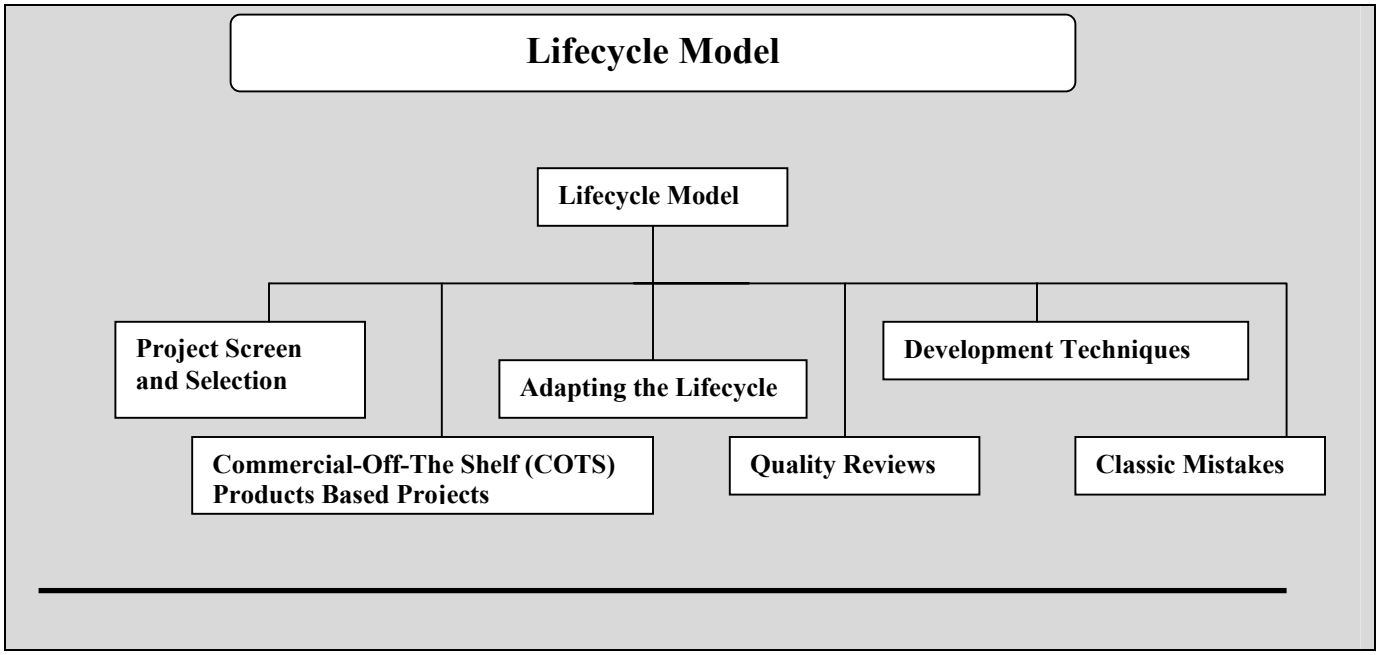
SECTION 2

LIFECYCLE MODEL

Table of Contents

Section 2: Lifecycle Model

Highlights of Phase



Section 2: Lifecycle Model

Overview

Description:

This section describes the lifecycle model used for the State of Michigan's SDLC. This model partitions the lifecycle into ten major phases, as shown in *Exhibit 2.0-1, Software Lifecycle Phases and Deliverables*. Each phase is divided into activities and tasks, and has a measurable end point (Phase Exit). The execution of all ten phases is based on the premise that the quality and success of a software product depends on a feasible concept, comprehensive and participatory project planning, commitments to resources and schedules, complete and accurate requirements, a sound design, consistent and maintainable programming techniques, and a comprehensive testing program. The lifecycle phases and activities are described in Sections 3 through 11. Appendixes A-T follow.

All maintenance projects should adapt the systems development lifecycle to meet the needs of the project. Any project that is defined as an emergency should follow the guidelines in Section 12, Emergency Maintenance.

Intermediate deliverables are produced during the performance of the activities and tasks in each phase. These deliverables are inspected and can be used to assess software integrity, quality, and project status. As a result, adequacy of requirements, correctness of designs, and quality of software products become known early in the effort.

At least one time for each deliverable, a Structured Walkthrough is performed. A Structured Walkthrough is an organized procedure for reviewing and discussing the technical aspects of software development deliverables including documentation. The walkthrough is usually conducted by a group of peers; however, it also includes reviewers outside the developer's immediate peer group *(see Appendix C).*

At least one time during each phase, an In-Phase Assessment is performed. An In-Phase assessment is an independent review of all deliverables during each lifecycle phase. A Quality Assurance representative typically conducts the assessment and the results are provided to the project manager. In-Phase Assessments are recommended after the achievement of all major project milestones and the completion of deliverables.

At the conclusion of each phase, a Phase Exit (see Appendix E) is initiated to review the deliverables of that phase and to determine whether to proceed to the next phase, continue work in the current phase, or abandon the project. The approval of the system owner and other project stakeholders at the conclusion of each phase enables both the system owner and the project manager to remain in control of the project throughout its life, and prevents the project from proceeding beyond authorized milestones.

The end products of the lifecycle are the software product, the data managed by the software, associated technical documentation, and customer training and support. The end products and services are maintained throughout the remainder of the lifecycle in accordance with documented configuration management procedures.

Overview

Description Continued:

The lifecycle model provides a method for performing the individual activities and tasks within an overall project framework. The phases and activities are designed to follow each other in an integrated fashion, whether the phases of development are accomplished sequentially, concurrently, or cyclically, etc. Project teams have the flexibility to adapt the lifecycle model to accommodate a particular development environment, systems development techniques (e.g., prototyping and rapid application development), or other project constraints.

The amount of project and system documentation required throughout the lifecycle depends on the size and scope of the project. System documentation needs to be at a level that allows for full system operability, usability, and maintainability. Typically, projects that require at least one work-year of effort should have a full complement of documentation. For projects that require less than one work-year of effort, the project manager and system owner should determine the documentation requirements. In addition, the project's security and quality assurance criteria may require the performance of other activities and the generation of additional documentation.

The requirements for documentation should not be interpreted as mandating formal; standalone printed documents in all cases. Progressive documents that continuously revise and expand existing documentation, online documents, forms, reports, electronic mail messages, and handwritten notes (e.g., informal conference records) are some examples of alternative documentation formats. Agencies should verify documentation standards within their sites. *(see Appendix J – Documentation Standards)*

The following sections provide additional information about the lifecycle model:

Project Screen and Selection
Adapting the Lifecycle
Development Techniques
Commercial-Off-The-Shelf (COTS) Products Based Projects
Quality Reviews
Classic Mistakes in Software Projects

Exhibit 2.0-1. Software Lifecycle Phases and Deliverables

Project Initiation Project Concept Document (PCD) Project Charter Project Feasibility **Planning** Project Plan Software Quality Assurance Plan Software Configuration Management Plan	**Requirements Definition** Requirements Traceability Matrix (draft) Continuity of Operations Statement/Plan Data Dictionary (draft) Software Requirements Specification Project Test Plan Acceptance Test Plan (draft) Project Plan (revised)
Functional Design Logical Model Data Dictionary (revised) Requirements Traceability Matrix (expanded) Functional Design Document Project Plan (revised)	**System Design** Data Dictionary (expanded) Physical Model Integration Test Plan (draft) System Test Plan (draft) Requirements Traceability Matrix (expanded) Conversion Plan System Design Document Program Specifications Programming Standards Project Plan (revised)
Programming Production Platform Acquisition Plan Installation Plan (draft) Requirements Traceability Matrix (expanded) Integration Test Plan (final) System Test Plan (final) Software Baseline Transition Plan Operating Documents (draft) Training Plan (draft) Project Plan (revised)	**Software Integration & Testing** Integration Test Reports System Test Report Operating Documents (final) Training Plan (final) Installation Plan (final) Acceptance Test Plan (final) Pre-acceptance Checklist Requirements Traceability Matrix (final) Maintenance Plan (draft) Project Plan (revised)
Installation and Acceptance Installation Test Materials Customer Training Materials Acceptance Test Materials Acceptance Test Report Acceptance Checklist Operational System Operating Documents Maintenance Plan (final) Project Plan (final)	**Closeout** Administrative Closure Financial Closure Project Audit **Emergency Maintenance** Revise all affected documentation
Retirement - Computer System Retirement Guidelines	

Note: A project may require other deliverables.

Criteria for Selecting a Development Project

Why Criteria are Needed:

New product development is a gamble, even under the best of circumstances. At one time it was appropriate to decide whether to go forward with a development project by "gut feel". Now, however, the marketplace and competitive situation has gotten so complicated and the cost of development projects has gotten so expensive that a more sophisticated means of evaluating and selecting projects is necessary.

You can improve the odds of success by selecting better projects and by weeding out, as early as possible, those ideas that make little sense to pursue. Listing and formalizing criteria for the evaluation and selection of development projects is a way of improving the odds. Seeing how well a project meets pre-established criteria is an important way of determining whether a project is worth starting or continuing.

Criteria are a qualitative and quantitative list of attributes that address marketing, financial, technological, manufacturing and competitive advantage issues. Well thought out criteria:

1. Provide guidelines that can be used to determine whether a project should be started.

2. Provide guidelines that can be applied at critical project milestones to determine whether a project should be continued.

3. Let people know how their ideas will be evaluated and help steer the types of ideas so that they are more likely to meet the needs of the company.

The criteria should be circulated to everyone in the company who has anything to do with new product development.

Elements of Criteria:

New product criteria address a number of important issues:

- ❑ Fit within the company's business and culture
- ❑ Compatibility with the company's core technologies
- ❑ Compatibility with the company's marketplace niche
- ❑ Ability to satisfy a specific customer need
- ❑ Possibility of achieving major market share
- ❑ Potential sales and profits
- ❑ Net Present Value
- ❑ Time for payback of development costs
- ❑ Cost of major tooling and machinery
- ❑ Internal rate of return on the investment
- ❑ Susceptibility to competitive attack
- ❑ Fit within the capabilities of existing and contemplated staff
- ❑ Growth potential of the product line
- ❑ Possibility of follow-on products
- ❑ Likelihood of being first in the marketplace with the new product
- ❑ Possibility of catching the competition by surprise
- ❑ Location of the market (local, national or international)
- ❑ Existence of a channel to the marketplace
- ❑ Potential downside risks of proceeding

Criteria for Selecting a Development Project

Elements of Criteria Continued:

- Potential risks of not proceeding
- Possible synergistic effect with current product lines
- Existence of identified lead customers
- Development cost and time
- Existence of a product champion
- Patentability and trade secrecy
- Likelihood that the new product will provide distinct competitive advantages to the company
- Availability of technology
- Resources required
- Ability to leverage available technology
- Urgency and criticality
- Uniqueness
- Technical merit
- Speed of entry into the marketplace
- Possibility of creating or dominating a niche market

Obviously, not all of these criteria are important for every company. Some are more important than others are. It is up to each agency to decide which of these must be met by a proposed project before it is allowed to go forward. The agency should divide its list of criteria into subcategories, those that are absolutely essential for all of its projects and those that are less important.

Also, different criteria should be met at different stages of development. During the concept evaluation stage, a project should be required to meet fewer criteria than would be the case at a later stage of development, when more significant funds would have to be committed if the project were to go forward.

Adopted from Dr. Philip A. Himmelfarb, President and Founder of Philip Adam & Associates.

Project Screen and Selection

Description:

The lifecycle model used in this systems development lifecycle can be applied to software projects of varying sizes. In this model, software projects are divided into three sizes: large, medium, and small. Each project size uses the same lifecycle phases. Medium and small projects may compress or combine phases and required documentation in direct proportion to the size of the development effort. The major differences between project sizes are determined by the following items:

- The estimated total labor hours (the level of effort) required to complete the project.
- The use of cutting edge or existing technology.
- The type and extent of both customer and system interface requirements.
- The project's contribution to, and impact on, the activities carried out by the customers and other agency organizations.

The requirements, constraints, and risks associated with the project also influence the determination of project size. The project size and any plans for adapting the lifecycle model are documented in the Project Plan, which is reviewed and approved by the system owner and other project stakeholders.

The following subsections provide descriptions of the three project sizes used in this lifecycle model. *Project Screen and Selection,* shows the level of effort and complexity measures used to define the three sizes.

The *Project Screen and Selection subsection was adopted from the Project Management Methodology desk reference guide.*

Project Screen and Selection

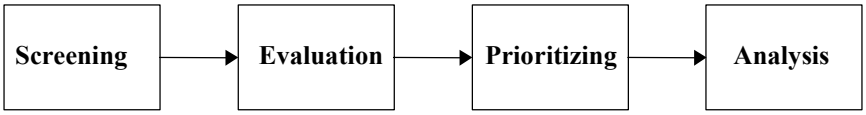
The Project Assessment Process:

Every project starts with an idea. That idea may be the result of a unique thought or design, it may respond to a regulatory mandate, it may answer a call for operational maintenance, or it may be as simple as providing scheduled updates. In essence, projects are generated for many different reasons; however, projects warrant special consideration for uniqueness, importance, cost, priority, and duration of effort. Accordingly, potential projects, so as not to under estimate their 'value-add' and timing, need to be subjected to an assessment process that will allow the sponsor, stakeholders, project team, and other interested parties to validate the potential project benefits and timing.

Because many teams are initiated without regard for need and feasibility, an assessment process that includes valuation criteria should be pursued in order to ascertain the merit of the project itself. Major component phases of the assessment process can include, but may not be limited to, the items noted in Figure 1:

Figure 1

Project Assessment Phases



The Project Assessment Process Continued:

Screening

Typically, the screening phase consists of collecting data to determine whether or not the project belongs to a particular agency or organization and for preparing inputs for the Evaluation Phase. The perceived urgency of implementing ideas as a project will determine the timing delay in preparing data for review. This phase of the effort should be a quick and inexpensive exercise.

Evaluation

The Evaluation Phase builds on information gathered in the Screening Phase and provides, in greater detail, potential project information that will be used for evaluation. This information is then used to make such determinations as whether or not the idea warrants a project effort, integrates into the agency strategy, fits within current budget constraints, and/or conflicts with ongoing projects. It will help detail the "protracted" benefits of the project. This phase may require the inputs of "outside" experts, the utilization of computational analysis, or it may include the use of technological forecasting. The results of the Evaluation Phase may indicate that the idea has reached an acceptable level to be considered a project. This would lead to the next step of prioritizing the implementation of this project with regard to the current agency workload.

Project Screen and Selection

The Project Assessment Process Continued:

Prioritizing

In the Prioritizing Phase, each idea (if there is more than one idea or if there is a comparison with ongoing projects) is weighted and appraised in terms of its relative strengths and weaknesses. This weighting would determine not only its individual merit as a project to pursue, but it would indicate a relative strength compared to ongoing or competing projects. In order to determine whether to pursue this project, a number of various techniques may be used. A few of the more generally accepted procedures are:

- Checklist/Scoring Models – a "spreadsheet" type analysis weighting various projects.
- Cost Benefit Analysis – a comparison of benefits from completing the project versus the outcomes of not instituting the project (this must be carefully considered when the benefits are difficult to measure; e.g., conducting a training seminar versus installing a "tele-file" system).
- Risk Analysis – an analysis of issues created while the potential project is being conceived. The intent of Risk Analysis is to try and quantify concerns that could possibly impede project progress and deter outcome. (A most popular and useful technique used in analysis of a system is the Failure Modes and Effects Analysis–FMEA.)
- Decision Trees (flow networks) – a method for depicting and facilitating the analysis of problems that involves sequential decisions and variable outcomes over time.

It is hoped that any, or all of these, techniques will be useful in determining the relative merit of projects. Summarily, the results of this Prioritizing Phase will lead to an initial allocation of resources (human, capital, financial) toward beginning the efforts of the project.

Analysis

Analysis of enterprise considerations defines the final phase of project assessment selection. If the results of the Evaluation Phase indicate that the project should replace an ongoing project, then an analysis will need to be conducted as to how to reallocate resources to the new project while an ongoing project is temporarily put on hold or perhaps terminated. The process of going through an Analysis Phase will be used only if projects are determined that they will be competing for the same resources.

Project Screen and Selection

An Assessment Matrix

An Assessment Matrix, as referenced in the Prioritization Phase, provides a method for making decisions among alternatives based on their key components and benefits. When a senior executive must choose between two or more options, an assessment aid will provide straightforward, quantitative information that can be easily and quickly used to support decisions. Figure 2 displays an example of a completed weighting, assessment method that may be used in conjunction with agency generated criteria (see Figure 3 as an example) in determining relative merits of projects.

Figure 2

Project	Resources	Duration	Risk	Cost	Rating
Project New	**3**	**3**	**5**	**3**	**14**
Project 1	**1**	**1**	**1**	**3**	**6**
Project 2	**3**	**1**	**3**	**3**	**10**
Project 3	**5**	**3**	**3**	**3**	**14**
Project 4	**3**	**5**	**2***	**5**	**15**

*** Arbitrary decision**

Figure 3

Project Size	Resources	Duration	Risk	Cost
Small = 1	<5	< 3 months	No impact	< $50K
Medium = 3	< 10	< 6 months	Impacts Divisions	< $250K
Large = 5	> 10	> 6 months	Impacts other Agencies	> $250K

Ranking

A simple Likert ranking scale (1, 3, or 5) can be easily applied to choosing how projects are prioritized and implemented. The following ranking scale applies to the example above:

- A score of 4 – 8 = a small project
- A score of 9 – 15 = a medium project
- A score of 16 and higher = a large project

Because different Agencies have different internal requirements, it is suggested that each Agency determine the best methodology for implementing an assessment scheme for their use.

When Not to Formalize a Project Effort

The formalization of project efforts is as unique as there are numbers of projects being undertaken, and agencies undertaking them. However, it is generally accepted best practice that the establishment of project activities (scope, plan, WBS, scheduling and other project components as described in this methodology) need not be formalized for efforts with less than three people whose duration does not exceed one month.

Essentially, it is recommended that an assessment approach be kept flexible enough so that the effort and results are consistent with the size and complexity of the alternatives being evaluated, life cycle phase, and level and type of review being supported.

Adapting the Lifecycle

Description:

The systems development lifecycle implements well-defined processes in a lifecycle model that can be adapted to meet the specific requirements or constraints of any software project. This section provides guidelines for adapting the lifecycle processes to fit the characteristics of the project. These guidelines help ensure that there is a common basis across all software projects for planning, implementing, tracking, and assuring the quality of the deliverables.

Adaptations:

The lifecycle model has built-in flexibility. All of the phases and activities can be adapted to any size and scope systems development project. The lifecycle can be successfully applied to systems development projects, software maintenance or enhancements, and customization of commercial software. The lifecycle is appropriate for all types of systems development applications including client/server, mainframe, emergency maintenance, enhancements, web development, procurement, and commercial off-the-shelf (COTS) projects. The project stakeholders or the requirements for reporting technical results in formal reports may dictate adaptations to the lifecycle.

The lifecycle can be compressed to satisfy the needs of a small project, expanded to include additional activities or deliverables for a large or complex project, or supplemented to accommodate additional requirements, (e.g., security requirements). Any modifications to the lifecycle should be consistent with the established activities, documentation, and quality standards included in the lifecycle. Project teams are encouraged to adapt the lifecycle as long as the fundamental software development objectives are retained, quality, CMM Level 2 and CMM Level 3 requirements are not compromised.

The following are some examples of lifecycle adaptations:

- Change the order in which lifecycle phases are performed.
- Schedule phases and activities in concurrent or sequential order.
- Repeat, merge, or eliminate phases, activities, or deliverables.
- Include additional activities, tasks, or deliverables in a phase.
- Change the sequence or implementation of lifecycle activities.
- Change the development schedule of the deliverables.
- Combine or expand activities and the timing of their execution.

The lifecycle forms the foundation for project planning, scheduling, risks management, and estimation. When a lifecycle phase, activity, or deliverable is adapted, the change must be identified, described, and justified in the Project Plan. The Project Plan is developed as a separate document and includes a description of the systems development lifecycle that is the organization's standard process.

Adapting the Lifecycle

Adaptations Continued:

Exhibit 2.2-1, Adapting the Lifecycle, shows how phases can be combined to accommodate different size projects and systems development techniques. *Notes* are provided throughout the lifecycle phases to identify activities that have built-in project adaptation strategies. Adaptations should not introduce an unacceptable level of risk and require the approval of the system owner and other project stakeholders.
When adapting the lifecycle model, care must be taken to avoid the following pitfalls:

- Incomplete and inadequate project planning.
- Incomplete and inadequate definition of project objectives and software requirements.
- Lack of a development lifecycle that is supported by systems development preferred practices and tools.
- Insufficient time allocated to complete design before coding is started.
- Not defining and meeting criteria for completing one software lifecycle phase before beginning the next.
- Compressing or eliminating testing activities to maintain an unrealistic schedule.

Illustrations:

The following are illustrations that can be used in the Project Plan to describe different types of lifecycle adaptations. This illustration shows a scenario where the Project Feasibility activity will not be conducted in the Project Initiation Phase:

A Project Feasibility Document will not be performed for this software project. The need for the product has been documented in several organizational reports and was included in the fiscal year long-range plans. The platform for the project is currently used for all applications owned by this organization. There are no known vendor packages that will satisfy the functional requirements described by the system owner.

The following illustration shows how deliverables from two different phases can be combined into one deliverable:

The Functional Design and System Design documents will be combined into one design document. A Phase Exit will be conducted when the design document is completed. To reduce the risk associated with combining the two documents, the project team will develop prototype screens and reports for review and approval by the system owner/customer(s).

Adapting the Lifecycle

Illustrations Continued:

The following illustration shows how the ten-lifecycle phases can be compressed into seven phases for a small project:

This project will require 10 staff months of effort to enhance an existing application. The ten phases in the lifecycle will be combined into six phases as follows: (1) Project Initiation (2) Planning, (3) Requirements and Design, (4) Programming and Testing, (5) Installation and Acceptance, and (6) Project Closeout.

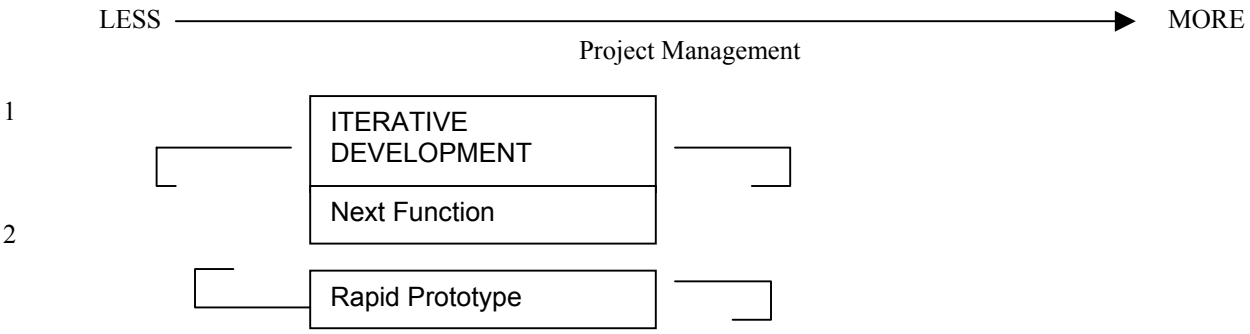
The following deviations will occur for document deliverables:

- *A Project Feasibility and Cost Benefit Analysis will not be necessary due to the restricted software and hardware platform.*
- *The Requirements Specification will be limited to the statement of enhancement requirements.*
- *The Functional Design and System Design documents will be combined into one design document.*
- *An amendment package will be developed for the existing Procedures Manual.*

Adapting the Lifecycle

Exhibit 2.2-1. Adapting the Lifecycle

Phases	Small	Medium	Large
Planning	X	X	X
Requirements Definition		X	X
Requirements Definition/Design	X		
Functional Design			X
System Design			X
System Design/Programming		X	
Programming			X
Programming/Testing	X		
Testing		X	X
Installation & Acceptance	X	X	X
Maintenance & Operations	X	X	X



Note: Iterative development and rapid prototyping are optional techniques that can be used on any size project.

1 Each iteration develops working function(s) from integrated program modules.

2 May develop any or all of requirements, system architecture, and system design.

Development Techniques

Description:

This section describes some examples of development techniques that can be used with the SOM Systems Development Lifecycle. The examples include high-level instructions on how to adapt the lifecycle phases to accommodate the development technique. *Exhibit 2.2-1, Adapting the Lifecycle,* shows how some development techniques can be used with the software lifecycle model. The examples provided here are not intended to be a comprehensive list of development techniques.

Segmented Development:

Segmented development is most often applied to large systems development projects where the project requirements can be divided into functional segments. Each segment becomes a separate project and provides a useful subset of the total capabilities of the full product. This segmentation serves two purposes: to break a large development effort into manageable pieces for easier project management and control; and to provide intermediate deliverables that form the building blocks for the complete product.

The lifecycle processes and activities are applied to each segment. The overall software objectives are defined, the system architecture is selected for the overall project, and a Project Plan for development of the first segment is written and approved by the system owner.

Segments are delivered to the system owner for evaluation or actual operation. The results of the evaluation or operation are then used to refine the content of the next segment. The next segment provides additional capabilities. This process is repeated until the entire software product has been developed. If significant problems are encountered with a segment, it may be necessary to reexamine and revise the project objectives, modify the system architecture, update the overall schedule, or change how the segments are divided.

Two major advantages of this approach are: the project manager can demonstrate concrete evidence that the final product will work as specified; and customers will have access to, and use of, segments or functions prior to the delivery of the entire software product.

Spiral Development:

Spiral development repeats the planning, requirements, and functional design phases in a succession of cycles in which the project's objectives are clarified, alternatives are defined, risks and constraints are identified, and a prototype is constructed. The prototype is evaluated and the next cycle is planned.

The project objectives, alternatives, constraints, and risks are refined based on this evaluation; then, an improved prototype is constructed. This process of refinement and prototyping is repeated as many times as necessary to provide an incrementally firm foundation on which to proceed with the project.

The lifecycle activities for the Planning, Requirements Definition, and Functional Design Phases are repeated in each cycle. Once the design is firm, the lifecycle phases for System Design, Programming, and Integration and Testing are followed to produce the final software product.

Development Techniques

Rapid Prototyping:

Rapid prototyping can be applied to any systems development lifecycle (e.g., segmented, spiral). Rapid prototyping is recommended for software development that is based on a new technology or evolutionary requirements.

With the rapid prototyping technique, the most important and critical software requirements are defined based on current knowledge and experience. A quick design addressing those requirements is prepared, and a prototype is coded and tested. The purpose of the prototype is to gain preliminary information about the total requirements and confidence in the correctness of the design approach. Characteristics needed in the final software product, such as efficiency, maintainability, capacity, and adaptability might be ignored in the prototype.

The prototype is evaluated, preferably with extensive customer participation, to refine the initial requirements and design. After confidence in the requirements and design approach is achieved, the final software is developed. The prototype might be discarded, or a portion of it used to develop the final product.

The normal systems development documentation requirements are usually postponed with prototyping efforts. Typically, the project team, project stakeholders, and system owner agree that the prototype will be replaced with the actual software product and required support documentation after proof of the model. The software that replaces the prototype should be developed using the lifecycle processes and activities.

Iterative Technique:

The iterative technique is normally used to develop software products piece by piece. Once the system architecture and functional or conceptual design are defined and approved, system functionality can be divided into logically related pieces called "drivers."

In iterative fashion, the project team performs system design, code, unit test, and integration test activities for each driver, thereby delivering a working function of the product. These working functions or pieces of the software product are designed to fit together as they are developed. This technique allows functions to be delivered incrementally for testing so that they can work in parallel with the project team. It also enables other functional areas, such as documentation and training, to begin performing their activities earlier and in a more parallel effort. In addition, the iterative technique enables progress to be visible earlier, and problems to be contained to a smaller scope.

With each iterative step of the development effort, the project team performs the lifecycle processes and activities.

Rapid Application Development:

Rapid Application Development (RAD) is a method for developing systems incrementally and delivering working pieces every 3 to 4 months, rather than waiting until the entire project is programmed before implementation. Over the years, many information projects failed because by the time the implementation took place, the business had changed.

Development Techniques

Rapid Application Development Continued:

RAD employs a variety of automated design and development tools, including Computer-Aided Software Engineering (CASE), fourth-generation language (4GLs), visual programming, and graphical user interface (GUI) builders, which get prototypes up and running quickly. RAD focuses on personnel management and customer involvement as much as on technology.

Joint Application Development:

Joint Application Development (JAD) is a RAD concept that involves cooperation between the designer of a computer system and the end user to develop a system that meets the customer's needs exactly. It complements other system analysis and design techniques by emphasizing participative development among system owners, customers, designers, and builders. During JAD sessions for system design, the system designer will take on the role of facilitator for possibly several full-day workshops intended to address different design issues and deliverables.

Object-Oriented Development:

Object-oriented development focuses on the design of software components that mimic the real world. A component that adequately mimics the real world is much more likely to be used and reused. The approach emphasizes how a system operates, as opposed to analysis, which is concerned with what a system is capable of doing. One of the most important advantages in using an object-oriented approach is the ability to reuse components. Traditional practices surrounding software development often mitigate against reuse. Short-term goals are stressed because today's milestones must be achieved before any thought can be given to milestones that may be months or years away. Borrowed or reused code is often code that has already been tested, and in the end, may translate into cost savings. Object-oriented development may make code reuse much easier but the amount of actual reuse may still depend on the motivation of the project managers, designers and programmers involved. Code reuse can also lead to faster software development. Object-oriented software is easier to maintain because its structure is inherently decoupled. This usually leads to fewer side effects when changes have to be made. In addition, object-oriented systems may be easier to adapt and scale (i.e., large systems can be created by assembling reusable subsystems).

Typically, the object-oriented process follows an evolutionary spiral that starts with customer communication, where the problem is defined. The technical work associated with the process follows the iterative path of analysis, design, programming, and testing. The fundamental core concepts in object-oriented design involve the elements of *classes, objects, and attributes.* Understanding the definition and relationships of these elements is crucial in the application of object-oriented technologies.

Development Techniques

Object-Oriented Development Continued:

It is recommend that the following object-oriented issues be well understood in order to form a knowledge base for the analysis, design, testing, and implementation of software using object-oriented techniques:

- What are the basic concepts and principles that are applicable to object-oriented thinking?
- How should object-oriented software projects be planned and managed?
- What is object-oriented analysis and how do its various models enable a software engineer to understand classes, their relationships and behavior?
- What is a 'use case' and how can it be applied to analyze the requirements of a system?
- How do conventional and object-oriented approaches differ?
- What are the components of an object-oriented design model?
- How are 'patterns' used in the creation of an object-oriented design?
- What are the basic concepts and principles that are applicable for testing of object-oriented software?
- How do testing strategies and test case design methods change when object-oriented software is considered?
- What technical metrics are available for assessing the quality of object-oriented software?

Software Procurement:

Software procurement process is the process by which software is acquired. The process is defined by the procurement plan, which identifies business and/or technical requirements development, purchase methodology, budget, resources for a Joint Evaluation Committee (executive and core team), high-level testing plan, required approvals, and contracting methods. The software may be an off-the-shelf package that is a project product, or it may be testing, quality assurance, operating system, or database management. (see *Project Management Methodology, Planning Phase)*

Deliverables:

The deliverables described in the SOM systems development lifecycle will be the same for much of the lifecycle and it is the responsibility of the project manager to adapt the deliverables accordingly and document adaptations in the Project Plan.

SDLC Reference:

Planning Phase, provides guidance for preparing a project plan. See the DIT Web site at: http://www.michigan.gov/dit.

Commercial Off-the-Shelf (COTS) Products Based Projects

Description:	There is a current trend in systems development to make greater use of Commercial-Off-The-Shelf (COTS) products, that is, to buy a ready-made system from a software manufacturer rather than developing it in-house from scratch. This carries with it a sense of getting a system that can do the job, at a reasonable cost, and getting new function in subsequent releases over time. A further breakdown of this is located in Appendix I.
SDLC References:	Appendix I – Commercial Off-the-Shelf Software.

Quality Reviews

Description:

Quality Reviews:

1) *Assure that the established system development and project management processes and procedures are being followed;*
2) *Exposures and risks to the current project plan are identified and addressed*

This section describes the quality review and assurance mechanisms that are used with the systems development lifecycle. The purpose of the quality reviews is to assure that the established system development and project management processes and procedures are being followed effectively, and that exposures and risks to the current project plan are identified and addressed. The quality reviews facilitate the early detection of problems that could affect the reliability, maintainability, availability, integrity, safety, security, or usability of the software product. The quality reviews enhance the quality of the end deliverables of a project.

The quality reviews are conducted as Peer Reviews, Structured Walkthroughs, In-Phase Assessments (IPA) and Phase Exits. The quality review used depends on the deliverable being reviewed, the point of time within the phase, and the role of the person conducting the review.

Peer Review:

A peer review is an informal review of systems development deliverables including documentation that can be conducted at any time at the discretion of the developer. The developers "peers"—frequently other developers working on the same project, perform these informal reviews. Informal reviews can be held with relatively little preparation and follow up activity. Review comments are informally collected and the product developer determines which comments require future action. Peer reviews focus on the specific content of a product and are geared to help the developer improve the product.

Some of the deliverables prepared are considered interim deliverables as they feed into a major deliverable or into another phase. The interim deliverables are ideal candidates for the peer review; however, all deliverables can be candidates for peer reviews. Frequent peer review should be conducted multiple times on a deliverable to ensure that it is free of defects.

Structured Walkthrough:

The structured walkthrough is an organized procedure for reviewing and discussing the technical aspects of systems development deliverables including documentation. Structured walkthroughs are used to find errors early in the development process and to improve the quality of the product. They are very successful in identifying design flaws, errors in analysis or requirements definition, and validating the accuracy and completeness of deliverables.

Structured walkthroughs are conducted during all phases of the project lifecycle. They are used during the development of work products that have been identified as having deliverables for each phase (*see Exhibit 2.0-1*), such as requirements, specifications, design, code, test data, and documentation. Structured walkthroughs are used after the deliverables have been completed to verify the correctness and the quality of the finished product. They should be scheduled in the work breakdown structure developed for the project plan and can be referred to as code reviews, design reviews, or inspections. Structured walkthroughs should also be scheduled to review small, meaningful pieces of work. The progress made in each lifecycle phase should determine the frequency of the walkthroughs;

however, they may be conducted multiple times on a deliverable to ensure that it is free of defects.

In-Phase Assessment:

The In-Phase Assessment (IPA) is a quality review that is conducted by a reviewer who is typically independent of the project. The reviewer assesses software development project's processes, work products, and deliverables to verify adherence to standards and that sound software development and project management practices are being followed. This is particularly important when multiple deliverables are developed in a single lifecycle phase. The reviewer assesses the deliverable and prepares an IPA report based on the information contained within the deliverable. An IPA does not require meetings among the involved parties to discuss the deliverable; however a meeting is often scheduled with the reviewer and the developer once the IPA report is completed in order to review the findings. Subject matter experts, such as documentation editors, may be used in addition to the assessor to further improve the quality of deliverables.

An IPA can be conducted anytime during a phase whenever a deliverable is stable enough, or near the end of a phase to prepare for phase exit. An IPA can be conducted for each of the deliverables or one IPA for multiple deliverables depending on when the work products are made available for review and the size of the deliverables. IPAs are conducted in all phases of the project lifecycle and should be scheduled in the work breakdown structure developed for the project plan. The IPA is described in detail in *Appendix D, In-Phase Assessment Process Guide*.

Phase Exit:

The phase exit ensures that a project conforms to the project management methodology, the systems development lifecycle, and that the State of Michigan's Enterprise IT Standards are identified. The phase exit is conducted by the project manager with the project stakeholders, e.g., system owner, customer point of contact, quality assurance point of contact, security point of contact, architecture and standards point of contact, project manager's supervisor, and platform point of contact. It is a high-level evaluation of all deliverables produced in a lifecycle phase. It is assumed that each deliverable has undergone several peer reviews and/or structured walkthroughs as appropriate and a successful IPA was conducted prior to the phase exit process. The phase exit focuses on the satisfaction of all requirements for the phase of the lifecycle, rather than the specific content of each deliverable.

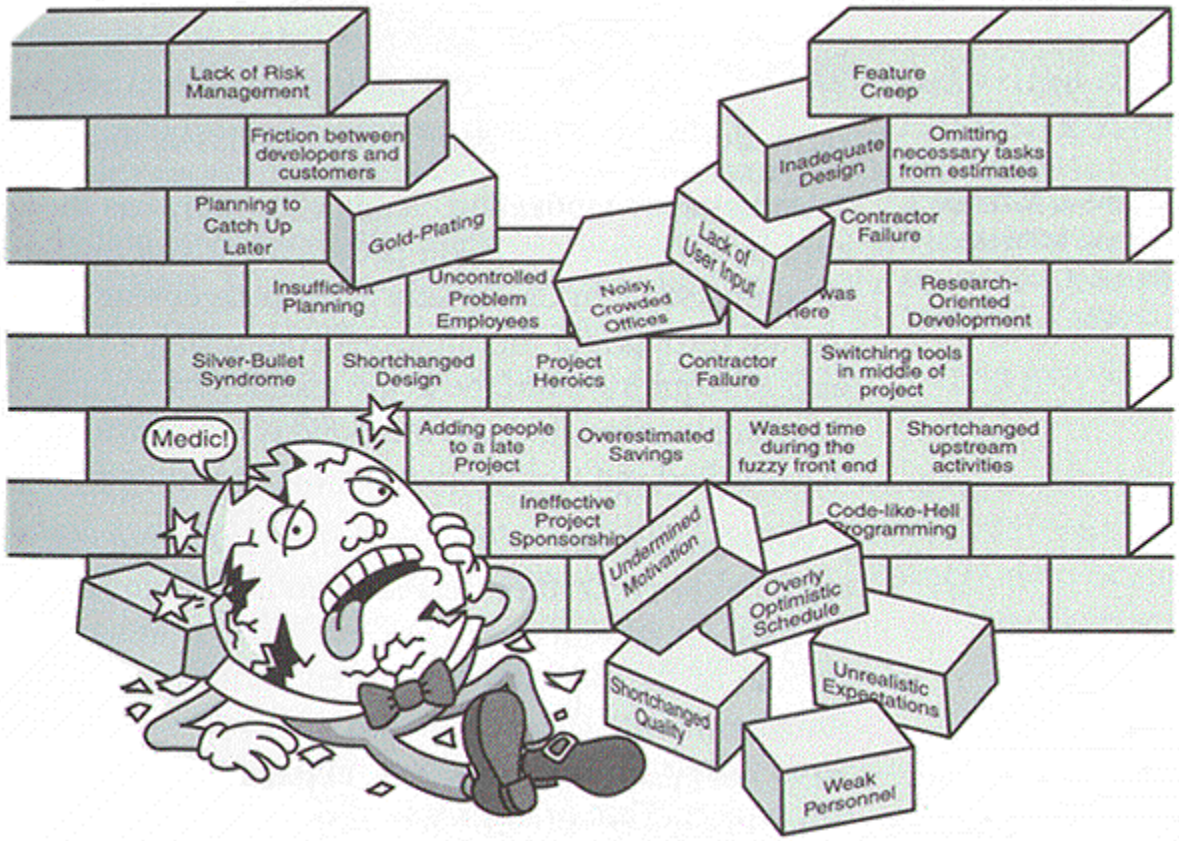
1) *Secure approval to continue with project:*
   - *Deliverables of phase;*
   - *Updated project plan;*
   - *All issues, concerns are closed*

2) *Document results*

The goal of a phase exit is to secure the concurrence (i.e., approval) of designated key individuals to continue with the project and to move forward into the next lifecycle phase. The concurrence is an approval (sign-off) of the deliverables for the current phase of development including the updated project plan. It indicates that all qualifications (issues and concerns) have been closed or have an acceptable plan for resolution. At a phase exit meeting, the project manager communicates the positions of the key personnel, along with qualifications raised during the phase exit process, issues that remain open from the IPA, and the action plan for resolution to the project team, stakeholders, and other interested meeting participants. The phase exit meeting is documented in summary form. Only one phase exit for

Phase Exit Continued:	each phase should be necessary to obtain approval assuming all deliverables have been accepted as identified in the project plan. The phase exit is described in detail in *Appendix E, Phase Exit Process Guide.*
SDLC References:	*Appendix C, Conducting Structured Walkthroughs*, provides a procedure and sample forms that can be used for structured walkthroughs. *Appendix D, In-Phase Assessment Process Guide*, provides a procedure and sample report form that can be used for In-Phase Assessments. *Appendix E, Phase Exit Process Guide*, provides a procedure and sample report form that can be used for phase exits.

Classic Mistakes in Software Projects



Classic Mistakes in Software Projects

Introduction:

Steve McConnel presents in his book Rapid Development a comprehensive list of "classic mistakes" in software projects (so called because they cause so predictable bad, yet common, results).

He divides them in four categories:

- People-Related Mistakes
- Process-Related Mistakes
- Product-Related Mistakes
- Technology-Related Mistakes

These mistakes will be discussed as a motivational factor for adopting effective development practices.

So many people, with such predictable, bad results that they deserve to be called “classic mistakes” have chosen some ineffective development practices so often. Most of the mistakes have a seductive appeal. Do you need to rescue a project that's behind schedule? Add more people! Do you want to reduce your schedule? Schedule more aggressively! Is one of your key contributors aggravating the rest of the team? Wait until the end of the project to fire him! Do you have a rush project to complete? Take whatever developers are available right now and get started as soon as possible!

Developers, managers, and customers usually have good reasons for making the decisions they do, and the seductive appeal of the classic mistakes is part of the reason these mistakes have been made so often. But because they have been made so many times, their consequences become easy to predict, and they rarely produce the results that people hope for.

This section enumerates three dozen classic mistakes. I have personally seen each of these mistakes made at least once, and I've made many of them myself. You'll recognize many of them from Case Study 3-1.

The common denominator in this list is that you won't necessarily get rapid development if you avoid the mistake, but you will definitely get slow development if you don't avoid it.

If some of these mistakes sound familiar, take heart. Many other people have made the same mistakes, and once you understand their effect on development speed you can use this list to help with your project planning and risk management.

Some of the more significant mistakes are discussed in their own sections in other parts of this book. Others are not discussed further. For ease of reference, the list has been divided along the development-speed dimensions of people, process, product, and technology.

Classic Mistakes in Software Projects

People:

Here are some of the people-related classic mistakes.

#1: Undermined motivation. Study after study has shown that motivation probably has a larger effect on productivity and quality than any other factor (Boehm 1981). In Case Study 3-1, management took steps that undermined morale throughout the project--from giving a hokey pep talk at the beginning to requiring overtime in the middle and going on a long vacation while the team worked through the holidays to providing bonuses that work out to less than a dollar per overtime hour at the end.

#2: Weak personnel. After motivation, either the individual capabilities of the team members or their relationship as a team probably has the greatest influence on productivity (Boehm 1981, Lakhanpal 1993). Hiring from the bottom of the barrel will threaten a rapid development effort. In the case study, personnel selections were made with an eye toward who could be hired fastest instead of who would get the most work done over the life of the project. That practice gets the project off to a quick start but doesn't set it up for rapid completion.

#3: Uncontrolled problem employees. Failure to deal with problem personnel also threatens development speed. This is a common problem and has been well understood at least since Gerald Weinberg published *Psychology of Computer Programming* in 1971. Failure to take action to deal with a problem employee is the most common complaint that team members have about their leaders (Larson and LaFasto 1989). In Case Study 3-1, the team knew that Chip was a bad apple, but the team lead didn't do anything about it. The result--redoing all of Chip's work--was predictable.

#4: Heroics. Some software developers place a high emphasis on project heroics, thinking that the certain kinds of heroics can be beneficial (Bach 1995). But I think that emphasizing heroics in any form usually does more harm than good. In the case study, mid-level management placed a higher premium on can-do attitudes than on steady and consistent progress and meaningful progress reporting. The result was a pattern of scheduling brinkmanship in which impending schedule slips weren't detected, acknowledged, or reported up the management chain until the last minute. A small development team held an entire company hostage because they wouldn't admit that they were having trouble meeting their schedule. An emphasis on heroics encourages extreme risk taking and discourages cooperation among the many stakeholders in the software-development process.

Some managers encourage this behavior when they focus too strongly on can-do attitudes. By elevating can-do attitudes above accurate-and-sometimes-gloomy status reporting, such project managers undercut their ability to take corrective action. They don't even know they need to take corrective action until the damage has been done. As Tom DeMarco says, can-do attitudes escalate minor setback into true disasters (DeMarco 1995).

Classic Mistakes in Software Projects

People Continued:

#5: Adding people to a late project. This is perhaps the most classic of the classic mistakes. When a project is behind, adding people can take more productivity away from existing team members than it adds through new ones. Fred Brooks likened adding people to a late project to pouring gasoline on a fire (Brooks 1975).

#6: Noisy, crowded offices. Most developers rate their working conditions as unsatisfactory. About 60 percent report that they are neither sufficiently quiet nor sufficiently private (DeMarco and Lister 1987). Workers who occupy quiet, private offices tend to perform significantly better than workers who occupy noisy, crowded work bays or cubicles. Noisy, crowded work environments lengthen development schedules.

#7: Friction between developers and customers. Friction between developers and customers can arise in several ways. Customers may feel that developers are not cooperative when they refuse to sign up for the development schedule that the customers want, or when they fail to deliver on their promises. Developers may feel that customers unreasonably insisting on unrealistic schedules or requirements changes after requirements have been baselined. There might simply be personality conflicts between the two groups.

The primary effect of this friction is poor communication, and the secondary effects of poor communication include poorly understood requirements, poor user-interface design, and, in the worst case, customers' refusing to accept the completed product. On average, friction between customers and software developers is so severe that both parties consider canceling the project (Jones 1994). Such friction is time-consuming to overcome, and it distracts both customers and developers from the real work of the project.

#8: Unrealistic expectations. One of the most common causes of friction between developers and their customers or managers is unrealistic expectations. In Case Study 3-1, Bill had no reason to think that the Giga-Quote program could be developed in six months except for the fact that the company needed it in that amount of time. Mike's failure to correct that unrealistic expectation was a major source of problems. In other cases, project managers or developers ask for trouble by getting funding based on overly optimistic schedule estimates. Sometimes they promise a pie-in-the-sky feature set. Although unrealistic expectations do not in themselves lengthen development schedules, they contribute to the perception that development schedules are too long, and that can be almost as bad. A Standish Group survey listed realistic expectations as one of the top five factors needed to ensure the success of an in-house business-software project (Standish Group 1994).

#9: Lack of effective project sponsorship. High-level project sponsorship is necessary to support many aspects of rapid development including realistic planning, change control, and the introduction of new development practices. Without an effective project sponsor, other high-level personnel in your organization can force you to accept unrealistic deadlines or make changes that undermine your project. Australian consultant Rob Thomsett argues that lack of an effective project sponsor virtually guarantees project failure (Thomsett 1995).

Classic Mistakes in Software Projects

People Continued:

#10: Lack of stakeholder buy-in. All of the major players in a software-development effort must buy in to the project. That includes the executive sponsor, team leader, team members, marketing, end-users, customers, and anyone else who has a stake in it. The close cooperation that occurs only when you have complete buy-in from all stakeholders allows for precise coordination of a rapid development effort that is impossible to attain without good buy-in.

#11: Lack of user input. The Standish Group survey found that the number one reason that IS projects succeed is because of user involvement (Standish Group 1994).

#12: Politics placed over substance. Larry Constantine reported on four teams that had four different kinds of political orientations (Constantine 1995a). "Politicians" specialized in "managing up," concentrating on relationships with their managers. "Researchers" concentrated on scouting out and gathering information. "Isolationists" kept to themselves, creating project boundaries that they kept closed to non-team members. "Generalists" did a little bit of everything: they tended their relationships with their managers, performed research and scouting activities, and coordinated with other teams through the course of their normal workflow. Constantine reported that initially the political and generalist teams were both well regarded by top management. But after a year and a half, the political team was ranked dead last. Putting politics over results is fatal to speed-oriented development.

#13: Wishful thinking. I am amazed at how many problems in software development boil down to wishful thinking. How many times have you heard statements like these:

> "None of the team members really believed that they could complete the project according to the schedule they were given, but they thought that maybe if everyone worked hard, and nothing went wrong, and they got a few lucky breaks, they just might be able to pull it off."
>
> "Our team hasn't done very much work to coordinate the interfaces among the different parts of the product, but we've all been in good communication about other things, and the interfaces are relatively simple, so it'll probably take only a day or two to shake out the bugs."
>
> "We know that we went with the low-ball contractor on the database subsystem and it was hard to see how they were going to complete the work with the staffing levels they specified in their proposal. They didn't have as much experience as some of the other contractors, but maybe they can make up in energy what they lack in experience. They'll probably deliver on time."
>
> "We don't need to show the final round of changes to the prototype to the customer. I'm sure we know what they want by now."
>
> "The team is saying that it will take an extraordinary effort to meet the deadline, and they missed their first milestone by a few days, but I think they can bring this one in on time."

Classic Mistakes in Software Projects

People Continued:

but I think they can bring this one in on time."

Wishful thinking isn't just optimism. It's closing your eyes and hoping something works when you have no reasonable basis for thinking it will. Wishful thinking at the beginning of a project leads to big blowups at the end of a project. It undermines meaningful planning and may be at the root of more software problems than all other causes combined.

Process:

Process-related mistakes slow down projects because they squander people's talents and efforts. Here are some of the worst process-related mistakes.

#14: Overly optimistic schedules. The challenges faced by someone building a three-month application are quite different than the challenges faced by someone building a one-year application. Setting an overly optimistic schedule sets a project up for failure by underscoping the project, undermining effective planning, and abbreviating critical upstream development activities such as requirements analysis and design. It also puts excessive pressure on developers, which hurts developer morale and productivity. This was a major source of problems in Case Study 3-1.

#15: Insufficient risk management. Some mistakes have been made often enough to be considered classics. Others are unique to specific projects. As with the classic mistakes, if you don't actively manage risks, only one thing has to go wrong to change your project from a rapid-development project to a slow-development one. Failure to manage risks is one of the most common classic mistakes.

#16: Contractor failure. Companies sometimes contract out pieces of a project when they are too rushed to do the work in-house. But contractors frequently deliver work that's late, that's of unacceptably low quality, or that fails to meet specifications (Boehm 1989). Risks such as unstable requirements or ill-defined interfaces can be magnified when you bring a contractor into the picture. If the contractor relationship isn't managed carefully, the use of contractors can slow a project down rather than speed it up.

#17: Insufficient planning. If you don't plan to achieve rapid development, you can't expect to achieve it.

#18: Abandonment of planning under pressure. Projects make plans and then routinely abandon them when they run into schedule trouble (Humphrey 1989). The problem isn't so much in abandoning the plan as in failing to create a substitute and then falling into code-and-fix mode instead. In Case Study 3-1, the team abandoned its plan after it missed its first delivery, and that's typical. The result was that work after that point was uncoordinated and awkward--to the point that Jill even started working on a project for her old group part of the time and no one even knew it.

Classic Mistakes in Software Projects

Process Continued:

#19: Wasted time during the fuzzy front end. The "fuzzy front end" is the time before the project starts, the time normally spent in the approval and budgeting process. It's not uncommon for a project to spend months or years in the fuzzy front end and then to come out of the gates with an aggressive schedule. It's much easier and cheaper and less risky to save a few weeks or months in the fuzzy front end than it is to compress a development schedule by the same amount.

#20: Shortchanged upstream activities. Projects that are in a hurry try to cut out nonessential activities, and since requirements analysis, architecture, and design don't directly produce code, they are easy targets. On one disaster project that I took over, I asked to see the design. The team lead told me, "We didn't have time to do a design."
Also known as "jumping into coding," the results of this mistake are all too predictable. In the case study, a design hack in the bar-chart report was substituted for quality design work. Before the product could be released, the hack work had to be thrown out and the higher quality work had to be done anyway. Projects that skimp on upstream activities typically have to do the same work downstream at anywhere from 10 to 100 times the cost of doing it properly in the first place (Fagan 1976; Boehm and Papaccio 1988). If you can't find the 5 extra hours to do the job right the first time, where are you going to find the 50 extra hours to do it right later?

#21: Inadequate design. A special case of shortchanging upstream activities is inadequate design. Rush projects undermine design by not allocating enough time for it and by creating a pressure-cooker environment that makes thoughtful consideration of design alternatives difficult. The design emphasis is on expediency rather than quality, so you tend to need several ultimately time-consuming design cycles before you finally complete the system.

#22: Shortchanged quality assurance. Projects that are in a hurry often cut corners by eliminating design and code reviews, eliminating test planning, and performing only perfunctory testing. In the case study, design reviews and code reviews were given short shrift in order to achieve a perceived schedule advantage. As it turned out, when the project reached its feature-complete milestone it was still too buggy to release for five more months. This result is typical. Short-cutting a day of QA activity early in the project is likely to cost you 3 to 10 days of activity downstream (Jones 1994). This inefficiency undermines development speed.

#23: Insufficient management controls. In the case study, there were few management controls in place to provide timely warnings of impending schedule slips, and the few controls there were in place at the beginning were abandoned once the project ran into trouble. Before you can keep a project on track, you have to be able to tell whether it's on track.

#24: Premature or too frequent convergence. Shortly before a product is scheduled to be released there is a push to prepare the product for release--improve the product's performance, print final documentation, incorporate final help-system hooks, polish the installation program, stub out functionality that's not going to be ready on time, and so on. On rush projects, there is a tendency to force convergence early. Since it's not

Classic Mistakes in Software Projects

Process Continued:

possible to force the product to converge when desired, some rapid development projects try to force convergence a half dozen times or more before they finally succeed. The extra convergence attempts don't benefit the product. They just waste time and prolong the schedule.

#25: Omitting necessary tasks from estimates. If people don't keep careful records of previous projects, they forget about the less visible tasks, but those tasks add up. Omitted effort often adds about 20 to 30 percent to a development schedule (van Genuchten 1991).

#26: Planning to catch up later. If you're working on a six-month project, and it takes you three months to meet your two-month milestone, what do you do? Many projects simply plan to catch up later, but they never do. You learn more about the product as you build it, including more about what it will take to build it. That learning needs to be reflected in the schedule. Another kind of reestimation mistake arises from product changes. If the product you're building changes, the amount of time you need to build it changes too. In Case Study 3-1, major requirements changed between the original proposal and the project start without any corresponding reestimation of schedule or resources. Piling on new features without adjusting the schedule guarantees that you will miss your deadline.

#27: Code-like-hell programming. Some organizations think that fast, loose, all-as-you-go coding is a route to rapid development. If the developers are sufficiently motivated, they reason, they can overcome any obstacles. For reasons that will become clear throughout this book, this is far from the truth. The entrepreneurial model is often a cover for the old code-and-fix paradigm combined with an ambitious schedule, and that combination almost never works. It's an example of two wrongs not making a right.

Product:

Here are some classic mistakes are related to the way the product is defined.

#28: Requirements gold-plating. Some projects have more requirements than they need right from the beginning. Performance is stated as a requirement more often than it needs to be, and that can unnecessarily lengthen a software schedule. Users tend to be less interested in complex features than marketing and development are, and complex features add disproportionately to a development schedule.

#29: Feature creep. Even if you're successful at avoiding requirements gold-plating, the average project experiences about a 25-percent change in requirement over its lifetime (Jones 1994). Such a change can produce at least a 25-percent addition to the software schedule, which can be fatal to a rapid development project.

#30: Developer gold-plating. Developers are fascinated by new technology and are sometimes anxious to try out new features of their language or environment or to create their own implementation of a slick feature they saw in another product--whether or not it's required in their product. The effort required to design, implement, test, document, and support features that are not required lengthens the schedule.

Classic Mistakes in Software Projects

Product Continued:

#31: Push me, pull me negotiation. One bizarre negotiating ploy occurs when a manager approves a schedule slip on a project that's progressing slower than expected and then adds completely new tasks after the schedule change. The underlying reason for this is hard to fathom because the manager who approves the schedule slip is implicitly acknowledging that the schedule was in error. But once the schedule has been corrected, the same person takes explicit action to make it wrong again. This can't help but undermine the schedule.

#32: Research-oriented development. Seymour Cray, the designer of the Cray supercomputers, says that he does not attempt to exceed engineering limits in more than two areas at a time because the risk of failure is too high (Gilb 1988). Many software projects could learn a lesson from Cray. If your project strains the limits of computer science by requiring the creation of new algorithms or new computing practices, you're not doing software development; you're doing software research. Software-development schedules are reasonably predictable; software research schedules are not even theoretically predictable.

If you have product goals that push the state of the art--algorithms, speed, memory usage, and so on--you should expect great uncertainty in your scheduling. If you're pushing the state of the art and you have any other weaknesses in your project--personnel shortages, personnel weaknesses, vague requirements, unstable interfaces with outside contractors--you can throw predictable scheduling out the window. If you want to advance the state of the art, by all means, do it. But don't expect to do it rapidly!

Technology:

The remaining classic mistakes have to do with the use and misuse of modern technology.

#33: Silver-bullet syndrome. In the case study there was too much reliance on the advertised benefits of previously unused technologies (report generator, object oriented design, and C++) and too little information about how well they would do in this particular development environment. When project teams latch onto a single new methodology or new technology and expect it to solve their schedule problems, they are inevitably disappointed (Jones 1994).

#34: Overestimated savings from new tools or methods. Organizations seldom improve their productivity in giant leaps, no matter how good or how many new tools or methods they adopt. Benefits of new practices are partially offset by the learning curves associated with them, and learning to use new practices to their maximum advantage takes time. New practices also entail new risks, which you're likely to discover only by using them. You are more likely to experience slow, steady improvement on the order of a few percent per project than you are to experience dramatic gains. The team in Case Study 3-1 should have planned on, at most, a 10-percent gain in productivity from the use of the new technologies instead of assuming that they would nearly double their productivity.

A special case of overestimated savings arises when projects reuse code from previous projects. This can be a very effective approach, but the time savings is rarely as dramatic as expected.

Classic Mistakes in Software Projects

Technology Continued:

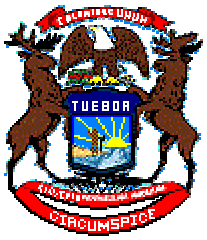
#35: Switching tools in the middle of a project. This is an old standby that hardly ever works. Sometimes it can make sense to upgrade incrementally within the same product line, from version 3 to version 3.1 or sometimes even to version 4. But the learning curve, rework, and inevitable mistakes made with a totally new tool usually cancel out any benefit when you're in the middle of a project.

#36: Lack of automated source-code control. Failure to use automated source-code control exposes projects to needless risks. Without it, if two developers are working on the same part of the program, they have to coordinate their work manually. They might agree to put the latest versions of each file into a master directory and to check with each other before copying files into that directory. But someone always overwrites someone else's work. People develop new code to out-of-date interfaces and then have to redesign their code when they discover that they were using the wrong version of the interface. Users report defects that you can't reproduce because you have no way to recreate the build they were using. On average, source code changes at a rate of about 10 percent per month, and manual source-code control can't keep up (Jones 1994).

Exhibit 2-0-3 Summary of Classic Mistakes contains a complete list of classic mistakes.

Exhibit 2.0-2 Summary of Classic Mistakes

People-Related Mistakes	Process-Related Mistakes	Product-Related Mistakes	Technology-Related Mistakes
1. Undermined motivation 2. Weak personnel 3. Uncontrolled problem employees 4. Heroics 5. Adding people to a late project 6. Noisy, crowded offices 7. Friction between developers and customers 8. Unrealistic expectations 9. Lack of effective project sponsorship 10. Lack of stakeholder buy-in 11. Lack of user input 12. Politics placed over substance 13. Wishful thinking	14. Overly optimistic schedules 16. Insufficient risk management 17. Contractor failure Insufficient planning 18. Abandonment of planning under pressure 19. Wasted time during the fuzzy front end 20. Shortchanged upstream activities 21. Inadequate design 22. Shortchanged quality assurance 23. Insufficient management controls 24. Premature or too frequent convergence 25. Omitting necessary tasks from estimates 26. Planning to catch up later 27. Code-like-hell programming	28. Requirements gold-plating 29. Feature creep 30. Developer gold-plating 31. Push me, pull me negotiation 32. Research-oriented development	33. Silver-bullet syndrome 34. Overestimated savings from new tools or methods 35. Switching tools in the middle of a project 36. Lack of automated source-code control



SYSTEMS DEVELOPMENT LIFECYCLE

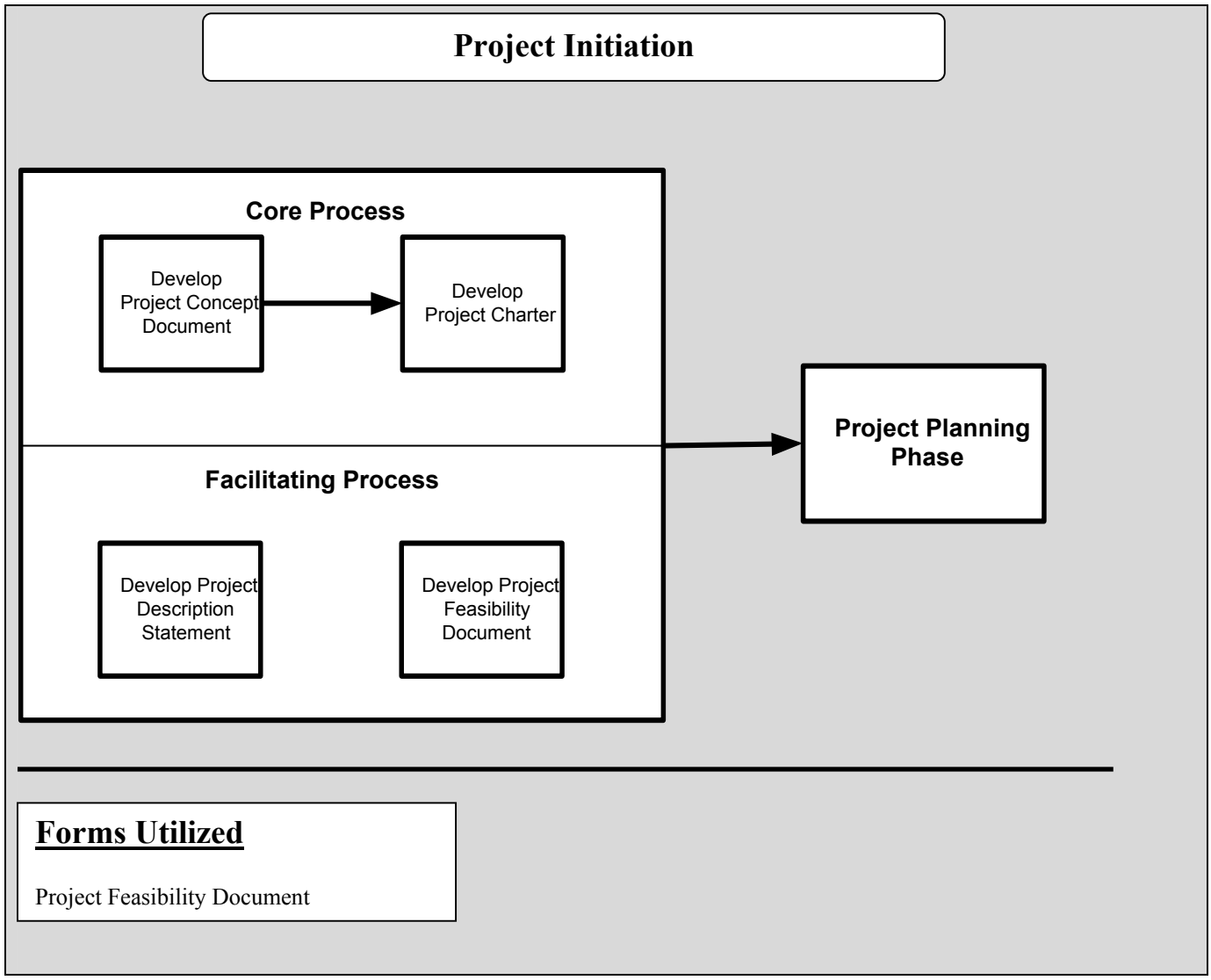
SECTION 3

PROJECT INITIATION

Section 3: Project Initiation Phase

Table of Contents

Highlights of Phase



Forms Utilized

Project Feasibility Document

Section 3: Project Initiation Phase

Overview

Description:

This section describes the project initiation activities associated with the systems development lifecycle. The project initiation phase is the first phase of the lifecycle. In this phase, the project is initiated by defining the activities associated with the project. The feasibility of the project is determined. A project feasibility document is developed. A project description statement is written. A project concept document is completed. A project appraisal package is prepared. A project charter document is written and approvals are obtained.

Approvals must be obtained from a steering committee on all documents in order to proceed with project.

Project initiation is the conceptual element of project management and systems development lifecycle. This section describes the basic documents that must be performed to get a systems development project started.

The purpose of the project initiation phase is to specify what the project should accomplish.

Resources:

The project concept, project charter, and project feasibility documents can be found on the project management methodology Web site at: http://www.michigan.gov/dit .

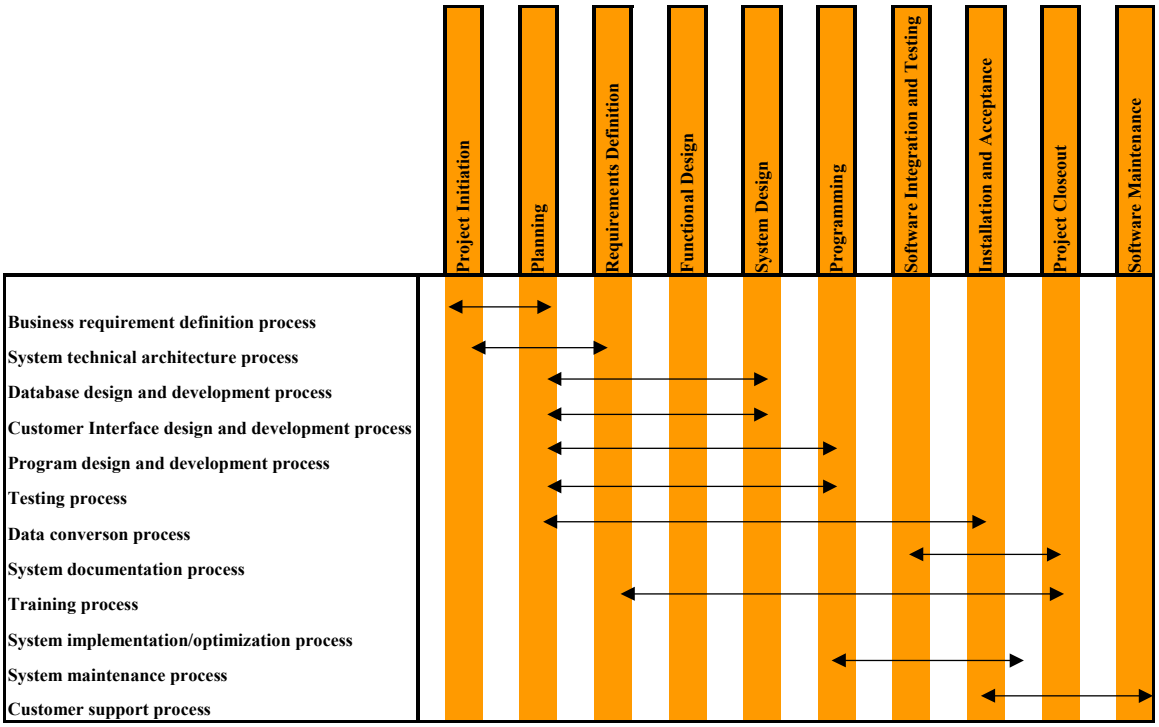
Transition to Planning Phase

Description:

The main function of a lifecycle model is to establish the order in which a project specifies, prototypes, designs, implements, reviews, tests, and performs its other activities. It establishes the criteria that you use to determine whether to proceed from one task to the next.

When the system approach is applied to the development of information system solutions, a multi-step process or cycle emerges. This is frequently called the **system development life cycle** (SDLC). Following figure summarizes what goes on in each phase of the SOM systems development lifecycle, which includes the steps of (1) project initiation, (2) planning, (3) requirements definition, (4) functional design, (5) system design, (6) programming, (7) software integration and testing, (8) installation and acceptance, (9) project closeout, and (10) emergency maintenance.

All of the activities involved are highly related and interdependent, Therefore, in actual practice, several developmental activities can occur at the same time. So, different parts of a development project can be at different phases of the development cycle. In addition, project may recycle back to repeat previous phases in order to modify and improve the results of an activity.



Taking Over an Existing Project

Become Familiar with Project:

a. Get a turnover briefing from the previous project manager.
b. Talk to the manager for the agency/division for business processes such as personnel management/staffing, financial management (budget planning, execution, and tracking), understand overhead issues.
c. Determine the current tasking
d. Learn the customer's detailed requirements, both technical and non-technical.
e. Determine what upper management expects of you.
f. Become familiar with the players and what their roles and responsibilities are:
   - ❑ Internal project groups/staff
   - ❑ Customer community
   - ❑ Sponsors
   - ❑ Contractors
   - ❑ Other interfacing groups
g. Determine the available internal resources.
   - ❑ People
   - ❑ Resources (equipment/software tools)
   - ❑ Facilities
h. Become familiar with the funding profile.
i. Determine what commitments have been made to/with any of the project's interfacing groups, customers, or other stakeholders.
j. Review any existing project plans (e.g., project plan, risk management plan, software quality assurance plan, configuration management plan, etc.) and project documentation (requirements specifications, design documents, test plans, test reports, etc.).
k. Become familiar with the reporting requirements. Learn what standards and processes are being applied. Determine what the current issues, politics, and "best practices" are – what is going right, what can be improved?
l. Become familiar with the metrics that are being collected and how they are reported and used.
m. Conduct a project status review.

Assume Management Responsibilities:

a. Become familiar with the State of Michigan's Systems Development Lifecycle (SDLC) and the Project Management Methodology (PMM).
b. Promote teamwork and set expectations
   - ❑ Have the team, including the project manager, review the SDLC and PMM documents
   - ❑ Attend any Project Management courses that will assist in the Process

c. Evaluate baseline/status and make changes as necessary using the Project Planning Process.
d. Report to the team and upper management changes resulting from applying the Project Planning Process.
e. Implement changes while continuing with the current tracking and oversight process. Review the SDLC and PMM and make changes / tailor as needed for project visibility and control.
f. Give upper management a full status brief.
g. Manage on-going activities as described in the SDLC and PMM documents.

Beginning a New Project

Get Oriented:

a. Talk to the manager for the agency/division for business processes such as personnel management/staffing, financial management (budget planning, execution, and tracking), understand overhead issues.
b. Identify the sponsor's tasking
c. Manage the customer's detailed requirements, both technical and non-technical.
d. Know what upper management expectations are.
e. Determine if there are any current stakeholders. If so, what are their roles and responsibilities?
   - ❑ Support groups such as contracts, QA, CM, and documentation support
   - ❑ Potential customer community
   - ❑ Contractors
   - ❑ Other interfacing groups
f. Determine required internal resources.
   - ❑ People
   - ❑ Resources (equipment/software tools)
   - ❑ Facilities
g. Determine the commitments that have been made to/with any of the players.
h. Become familiar with the reporting requirements. Establish standards and processes that are to be applied.

Assume Management Responsibility:

a. Become familiar with the State of Michigan's Systems Development Lifecycle (SDLC) and the Project Management Methodology (PMM).
b. Tailor SDLC and PMM to meet project goals.
c. Acquire necessary software tools.
d. Equip spaces with necessary hardware and software, furnishings
e. Promote teamwork and set expectations.
   - ❑ Have the team, including the project manager, review the SDLC and PMM documents
   - ❑ Attend any Project Management courses that will assist in the Process
f. Give upper management periodic (e.g., monthly) full status briefs.
g. Manage on-going activities as described in the SDLC and PMM documents.

How to Determine Project Status/Health

Description:

This section has been developed to assist software project managers and upper-level executives in the management of software projects. The objective is to guide managers in the proper supervision practices that will result in the delivery of quality products and services within the desired schedule and budget. With these goals in mind, this section includes the following:

- Questions a manager should ask to determine the status and health of a software project
- Characteristics of common software project problems and how to troubleshoot or (preferably) avoid them
- Brief summaries of essential SDLC processes
- Checklists to ensure successful completion of the phases of the software development effort
- Guidelines for project reviews and meetings, and checklists for the major management reviews)
- Metrics that managers use to measure the status of software projects and the software processes used

Questions to Determine Project Status/Health:

What are the vision, mission, goals, and objectives of the project?
How do you plan the activities on the project?
How do you know you are within budget & schedule?
What are the risks on this project?
How are the changes to the software handled?
How do you ensure a quality product?
How do you manage requirements?
How do you know the sponsor and user is satisfied with our work?
What training have the contractor and Government employees on the project had to do their tasks?
How do you perform contractor management /monitoring (if applicable)?
How do you estimate software?

Troubleshooting and Problem Avoidance:

Once you have asked questions to determine the project status and health, you might realize that you are in trouble. This section will help you to troubleshoot your problems. This section also will help you to avoid the problem later in your project or during your next project.

Systems Development Lifecycle Summaries:

Purpose: This section is intended to provide the manager with a quick reference about the systems development lifecycle process.

Definition: Process - a particular method of doing something, generally involving a number of steps or operations. Webster's World Dictionary

Each lifecycle is organized for easy reference as follows:

Project Initiation
Planning
Requirements Definition
Functional Design
Systems Design
Programming
Software Integration and Testing
Installation and Acceptance
Emergency Maintenance
Appendix

How to Determine Project Status/Health

Checklists and Templates:

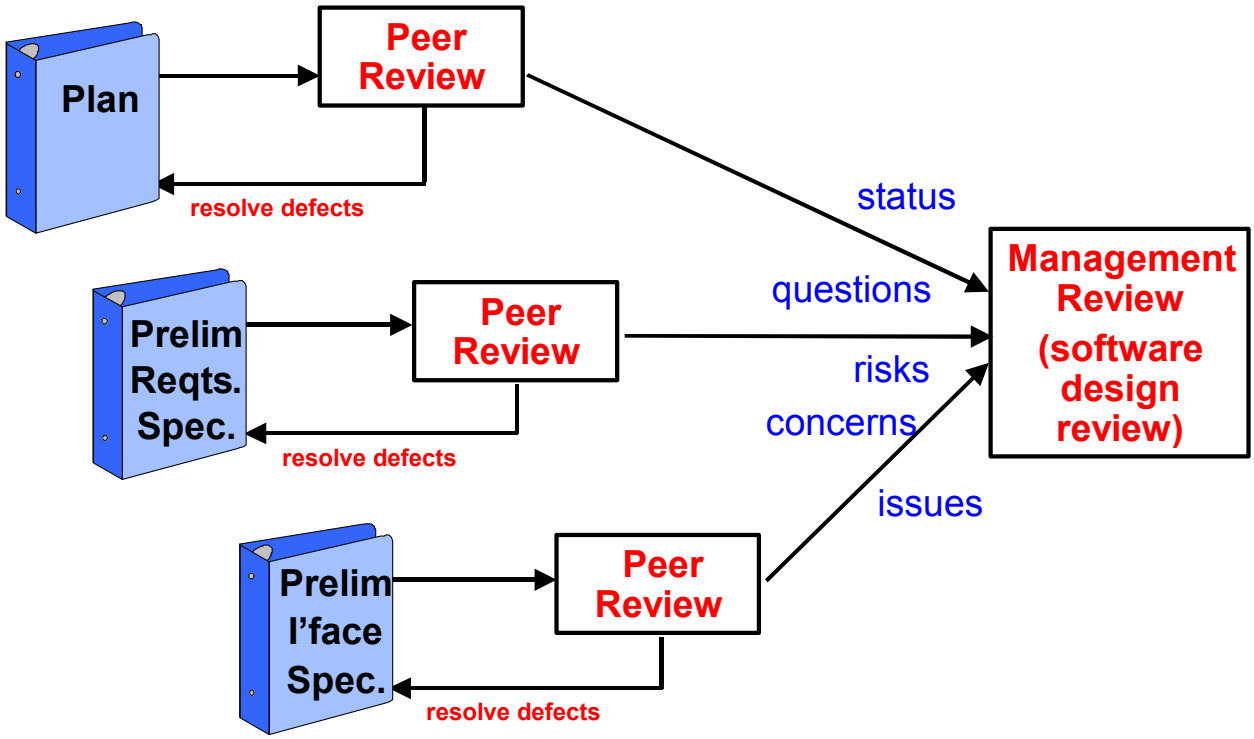
To provide a manager with an overview checklist or template of the activities performed by the Development Team during the initiation of a software development effort.

The checklists and templates are offered as guidelines for the activities to be covered during different phases of a software life cycle.

Note: These checklists and templates should be tailored for your specific project!

Project Reviews:

Purpose: To provide immediate technical feedback (including open issues and defects) to the developers to help them improve the product. These reviews deal only with technical issues. These reviews also provide feedback to management on the actual technical status of the project.



How to Determine Project Status/Health

Metrics:

Managers need the right information to make informed decisions. Used properly, metrics are a valuable source of that information. An old adage proffers that "you can't manage what you can't measure."
(note: the words "measurement" and "metric" are used synonymously)

Goal Based Measurements:

Managers should choose, collect, track, analyze, and make decisions based on measures that will show progress toward that manager's needs.

Measures should be chosen based on the goals the manager needs to track. Managers should be tracking progress to achieve the development goals of the project:

- Achieve the systems development lifecycle and project management capability defined through CMM Level 3 milestone.
- Produce quality software in shorter development cycles
- Reduce the cost of producing software throughout the life cycle
- Rapidly introduce new technology into the product and the software development process
- Integrate software across traditional system boundaries to provide a composite set of capabilities to the end user
- Continuously improve customer satisfaction

Managers are concerned that Agency goals are being met by tracking that:

- All projects have met the Sponsor's needs
- All projects have stable, educated staffs
- All projects have adequate resources
- All projects are contributing to the development goals
- All projects are improving their performance

Project managers should use measures that will relay information that the manager and the project has:

- Informed sponsors
- Realistic planning and budgeting
- Objective project insight
- Requirements stability
- Adequate staffing and computer resources
- On-target cost and schedule performance
- High Product Quality
- Contributions to the development goals
- Improved performance

Guidelines on Using Metrics:

Advice on implementing a metrics program suggests that a manager:

- Start small and only collect a few of the most relevant metrics first.
- Have a reason for the metric. For instance, analyzing the increased amount of sick leave, overtime, or turnover of project personnel over time may show a trend toward increased low morale, burnout, stress, and negative schedule impacts. A low number of customer complaints may indicate good, and open, communication between the developer, sponsor, and customer. Know when you are encountering identified project risks.

How to Determine Project Status/Health

Guidelines on Using Metrics (cont):

- Use the metrics collected. Project personnel must easily see the reason for the metric being collected, they must see the manager using the metrics for improvement of their project. History has shown that project personnel will not willingly provide metrics that have no apparent use.
- Collect similar metrics across projects to show larger trends within a Agency, Division, or organization. This also allows easier transfer of personnel among projects as expectations of them remain constant.
- Ask contractors for metrics via their status report, such as a task order log; progress reports, vouchers, and deliverables tracking; planned vs. actuals for staffing by skill levels, hours, dollars, schedules, and size; and tracking of open vs. closed action items, issues, and problems.
- Don't use metrics to measure individuals, use metrics to measure progress and performance of your project.

Project Metrics:

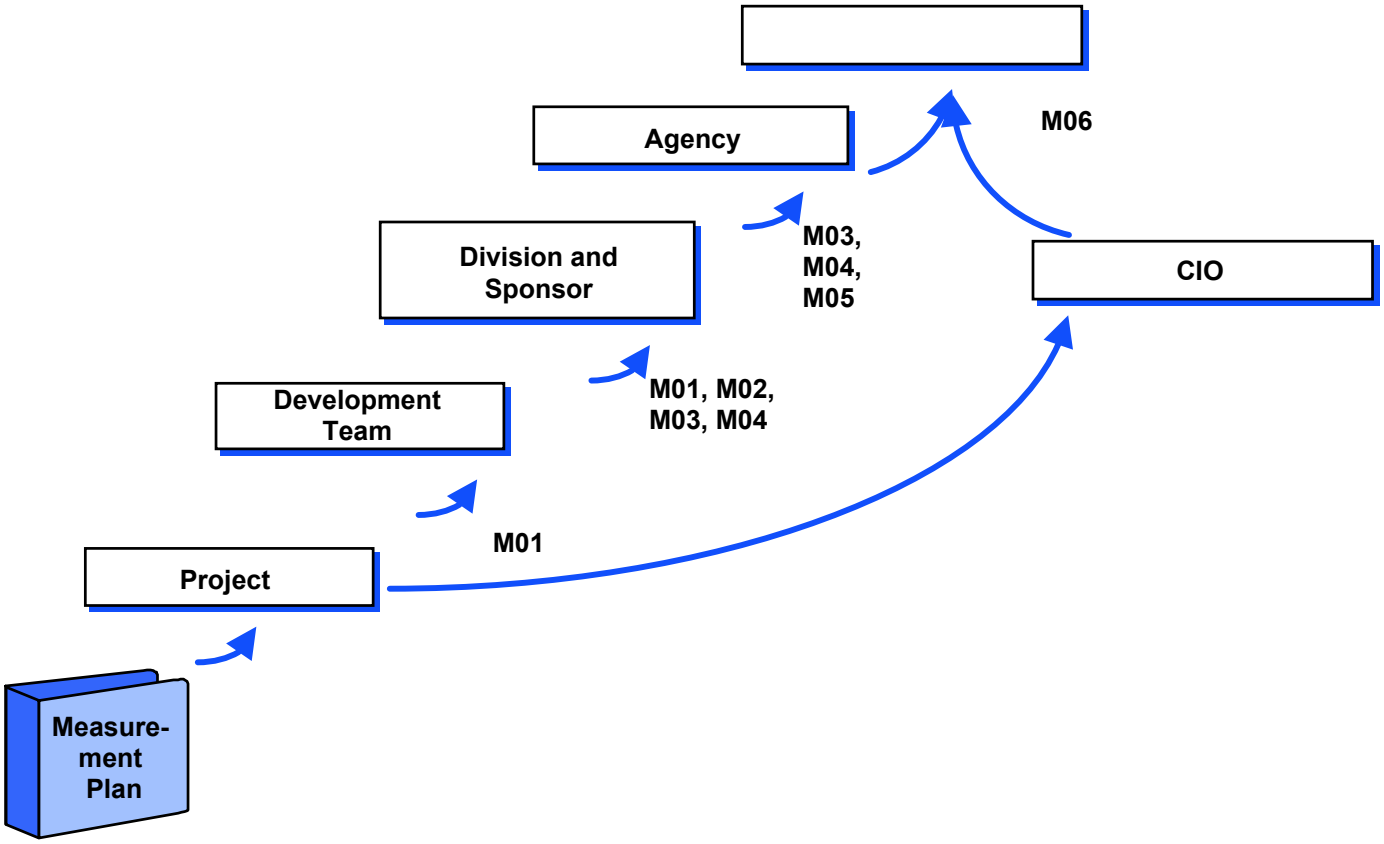
Management of software projects involves tracking and reviewing the software accomplishments and results against the plan and taking corrective action as necessary. These actions may include revising the software development plan to reflect the actual accomplishments, re-planning the remaining work, and/or taking actions to improve the performance of the project. The purpose of software project tracking and oversight is to establish adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.

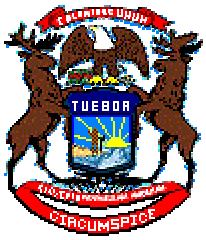
The goals of software project tracking and oversight are:

- Actual results and performances are tracked against the software plans
- Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans
- Changes to software commitments are agreed to by the affected groups and individuals.

A sample measurement plan is proposed below. Additional information about project status / health is available on the Research and Policy Web Site at: http://www.michigan.gov/dit .

How to Determine Project Status/Health





SYSTEMS DEVELOPMENT LIFECYCLE

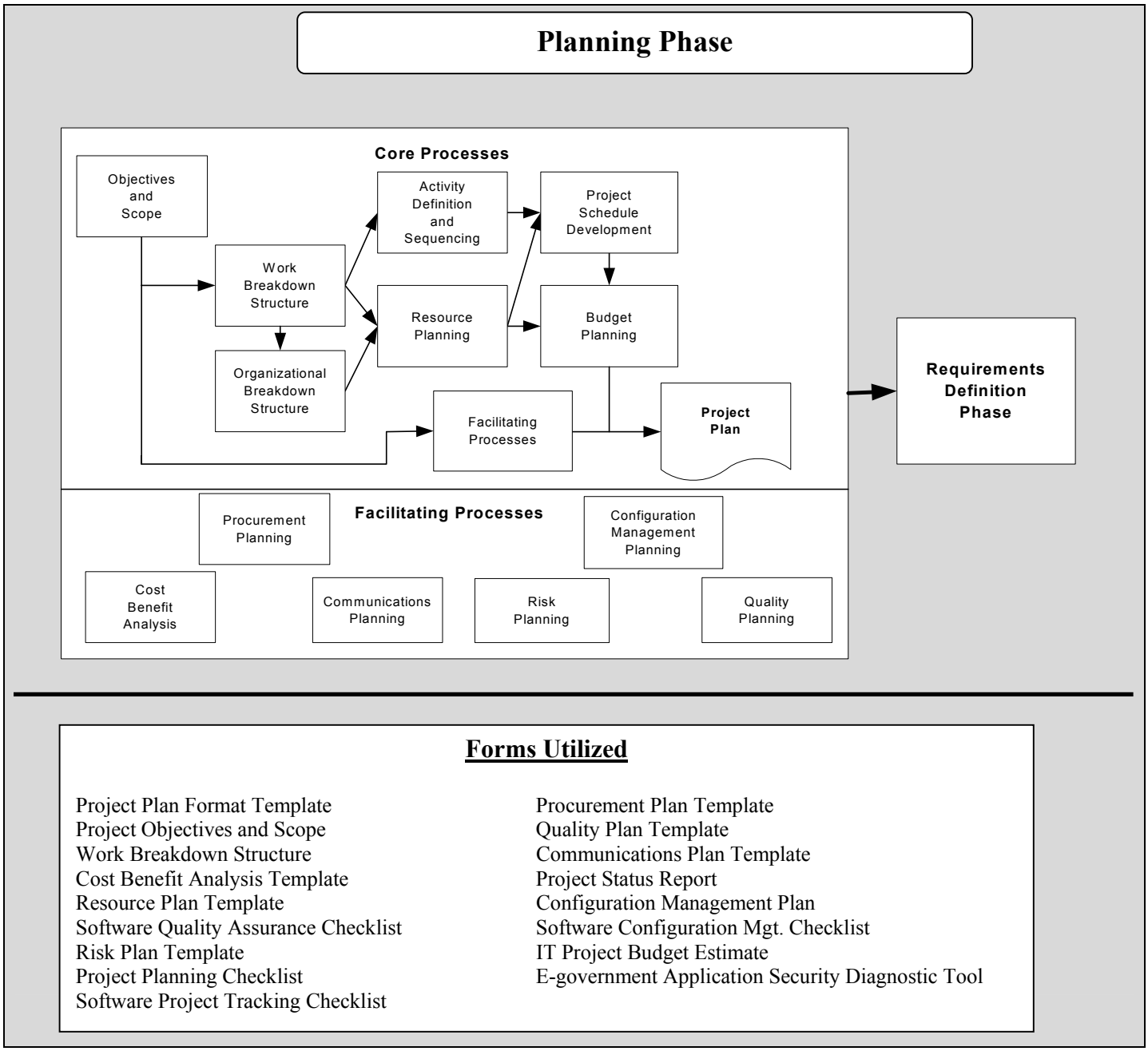
SECTION 4

PLANNING PHASE

Section 4: Planning Phase

Table of Contents

Highlights of Phase



Forms Utilized

Project Plan Format Template
Project Objectives and Scope
Work Breakdown Structure
Cost Benefit Analysis Template
Resource Plan Template
Software Quality Assurance Checklist
Risk Plan Template
Project Planning Checklist
Software Project Tracking Checklist
Procurement Plan Template
Quality Plan Template
Communications Plan Template
Project Status Report
Configuration Management Plan
Software Configuration Mgt. Checklist
IT Project Budget Estimate
E-government Application Security Diagnostic Tool

Section 4: Planning Phase

Overview

Description:

1) Identify system requirements;
2) Define project objectives and scope;
3) Estimate high-level project and functional requirements;
4) *Develop project plan, software quality assurance plan and software configuration management plan*

The Planning phase is the second phase in the systems development lifecycle. In this phase, the system requirements are identified, the customers' environment is analyzed, the project objectives and scope are defined, the high-level project and functional requirements are estimated, and the Project Plan, Risk Plan, Communications Plan, Budget for Project, Software Quality Assurance Plan and Software Configuration Management Plan are developed and approved.

The activities are performed to define and plan all aspects of the project and its work. Many of these initial planning activities are incorporated within the overall process.

Project planning applies to all projects regardless of their size. Planning involves selecting the strategies, policies, and procedures for achieving the objectives and goals of the project. Planning is deciding, in advance, what to do, how to do it, when to do it, where to do it, and who is going to do it.

The requirements identified in project related materials, e.g., a feasibility document, are the primary inputs to the Project Plan. The level of detail will vary depending on project size. The preparation of the Project Plan and related materials involve several critical planning issues such as the identification of preliminary requirements; staff, schedule, and cost estimates; technical and managerial approaches that will be used; and assessment of potential risks associated with the project. This information forms the foundation for all subsequent planning activities.

During this phase, the system owner and customers are interviewed to: identify their business needs and expectations for the product; gain a common understanding of the task assignment; and determine how the project supports the agency and organizational missions and long-range information resource management plans. The system owner is the organizational unit that is funding the project, and customers are the resources who will use the product.

In this phase, the project team should be focused on identifying what the project will automate, and whether developing an automated solution makes sense from business, cost, and technical perspectives. If the project is feasible, then time, cost, and resource estimates must be formulated for the project, and risk factors must be assessed. It is important for the project team to work closely with representatives from all functional areas that will be involved in providing resources, information, or support services for the project. The information that is gathered in this phase is used to plan and manage the project throughout its lifecycle.

This phase involves development of a Software Configuration Management Plan to track and control deliverables and a Software Quality Assurance Plan to assure the production and operation of high quality products on schedule, within budget, and within the specification of requirements (constraints) specified by the system owner and customer.

Section 4: Planning Phase

Overview

Review Process:

Quality reviews are necessary during this phase to validate the product and associated deliverables. The activities that are appropriate for quality reviews are identified in this section and *Section 2 Lifecycle Model*. The time and resources needed to conduct the quality reviews should be reflected in the project resources, schedule, and work breakdown structure.

SDLC References:

Section 2 Lifecycle Model, Quality Reviews, provides an overview of the Quality Reviews to be conducted on a project.

Appendix C, Conducting Structured Walkthroughs, provides a procedure and sample forms that can be used for structured walkthroughs.

Appendix D, In-Phase Assessment Process Guide, provides a procedure and sample report form that can be used for in-phase assessments.

Appendix E, Phase Exit Process Guide, provides a procedure and sample report form that can be used for phase exits.

SDLC and PMM

Description:

This section describes the project planning activities associated with the systems development lifecycle. The planning phase is performed once the project concept document, project charter and feasibility document have been met and the customer has accepted the project's product.

The systems development lifecycle recommends following the planning phase and the associated checklists and templates from the Project Management Methodology. These can be found on the Web site at: http://www.michigan.gov/dit .

Additional SDLC Planning Activities

Responsibility:

Project Manager/Team

Description:

In addition to the Project Management Methodology (PMM) activities, the feasibility of successfully developing and implementing the project is determined. Software and hardware alternatives are reviewed and used to formulate preliminary platform options. Project feasibility leads to a "go" or "no go" decision about the project. Determining project feasibility is an interactive process of collecting and analyzing data and searching for cost-effective, viable technical solutions.

1) *Determine feasibility of successfully developing and implementing the project;*
2) *Review software and hardware alternatives;*
3) *Make "go" or "no go" decision;*
4) *Conduct research;*
5) *Complete project feasibility*

Use the project objectives, scope, and high-level requirements as the basis for determining project feasibility. Work with the customer organization and functional area representatives to address technical issues and risks. Conduct research and investigate documents and other resources. When determining project feasibility, a brief feasibility review may be all that is required; however, when a proposed solution is mandated to define risks and challenges to successfully complete a project, an in depth project feasibility with a detailed analysis of benefits and costs may be required.

Note: *Feasibility may not be an issue for some small software development projects. A Feasibility Review is not required when feasibility is obvious.*

Sample Questions:

The following is a list of sample questions that can be used to help determine the feasibility of a project:

- ❑ Can the customer's needs/problems best be satisfied with a manual process, automated process, or combination?
- ❑ Is it cost-effective to develop an automated process?
- ❑ Is the scope of the project feasible within time, resource, and hardware and software constraints and limitations?
- ❑ Is there at least one technically feasible automated solution for the project?
  - If a project is well defined and has no automation issues, a single straightforward automated solution may sufficiently demonstrate cost and technical feasibility.
  - Where automation issues have been identified, technical alternatives should be associated with each proposed solution.

Deliverables:

Refer to each task for applicable deliverables.

Review Process:

Refer to each task for applicable review processes.

Tasks:

The following tasks are involved in determining project feasibility:

Investigate Software Alternatives
Investigate Hardware Alternatives
Formulate Platform Options
Conduct Feasibility Review (PMM)
Conduct Cost Benefit Analysis (PMM)
Conduct Project Feasibility (PMM)

Establish Communication With Plan

Responsibility:	Project Manager
Description:	Early contact with the functional areas (e.g., operations, security, finance, documentation, oversight activities, etc.) that will provide input to, or support for, the project is necessary for developing accurate estimates of the project scope, cost, resources, and schedule. Representatives of these functional areas should be involved in all phases of the project lifecycle and are participants in the Phase Exit process.
1) Profile software project; 2) Complete project planning questionnaire; 3) List all functional areas; 4) Send each points-of-contact list; 5) Request information from recipients	Develop a brief profile about the software project. Provide enough information so that the points-of-contact in each functional area will be able to estimate support requirements and resource allocations for the project. A sample project planning questionnaire that can be used for creating the project profile is provided at the end of this section. Develop a list of all functional areas and points-of-contact who will provide input to, or support the project. Send each point-of-contact the project profile and request input from all recipients. **Note:** This activity is not necessary for small software development projects that do not require input from other functional areas.
SDLC Reference:	Information on the identification of functional areas can be found in *Appendix E, Phase Exit Process Guide*, the Project Plan templates and the Project Planning Questionnaire can be found on the Research and Policy Web site at: http://www.michigan.gov/dit.
Deliverables:	Place a copy of the project profile in the Project Notebook. Update the project profile as needed to maintain an accurate description of the software product. Keep the list of functional area contacts current and maintain a copy in the Project Notebook and the Project Plan. Use this list as the starting point when functional areas need to be contacted about involvement in project activities such as Phase Exits.
Review Process:	A peer review of the Project Planning Questionnaire is optional; however, it is encouraged as it may provide helpful insight.
Project Planning Questionnaire:	Enable project teams (immediate and extended) to be cognizant of the disparate planning activities which can affect project outcome. Provide early notification to the stakeholders that a new project may involve their area, and information to help plan resource estimates and identify risks. See project questionnaire on the Research and Policy Web site at: http://www.michigan.gov/dit.

Develop Project Plan

Responsibility: Project Manager

Description:

The purpose of the Project Plan (sometimes called a Software Development Plan) is to establish reasonable plans for performing the systems development activities
and for managing and tracking the software project. The following project management activities and lifecycle activities described in this phase provide input for the Project Plan. The project management activities are initiated in the early phases of, and in parallel with the overall project planning:

- ❑ Define the management approach for the project including project tracking and oversight activities.
- ❑ Formulate the technical approach for the project.
- ❑ Determine the project standards to be used in developing the plan, e.g., enterprise standards, project standards, approved statements of work, etc., and determine the selected procedures, methods and project standards for developing and maintaining software.
- ❑ Collect the information needed to develop the project estimates and assess their reasonableness.
- ❑ Prepare an estimate of capacity requirements for the projects systems development facilities and support tools. Estimates of capacity requirements for facilities and support tools that are based on the size estimates of the deliverables and other characteristics. Examples of development facilities and support tools include:

  - Development computers and peripherals
  - Test computers and peripherals
  - Target computer environment
  - Other support software

Responsibilities are assigned and commitments are negotiated to procure or develop these facilities and support tools. The support efforts are budgeted and agreements (charters) are documented. All effected groups review plans.

- ❑ Establish the project development team, i.e., quality assurance, configuration management and documentation support, and coordinate and negotiate Plans with the project team.
- ❑ Establish the project schedule, i.e., a lifecycle with predefined phases of manageable size is defined.

The Project Plan is managed and controlled. The plan is reviewed by:

- ❑ Project Manager
- ❑ Project Software Manager
- ❑ Other Software Managers
- ❑ Other affected groups

Develop Project Plan

Note: A Project Plan is an effective management tool that is recommended for all projects regardless of size. The lifecycle phases documented in the plan can be consolidated for small projects.

SDLC Reference:

Section 2 Lifecycle Model, 2.2 Adapting the Lifecycle, discusses lifecycle issues in relation to developing a Project Plan.

Resources:

The following resources are available to assist with the development of a Project Plan:

- A Project Plan Document that provides a plan for a fictitious project
- Several project models were created in Microsoft Project 98 and NIKU to support the development of the project schedule. The project models provide detailed work breakdown structures for (1) large project model, (2) software procurement model, (3) static web page model. (4) internet/intranet development and many more.

The Institute of Electrical and Electronics Engineers (IEEE) Standard for Software Project Management Plans (Std 1058.1-1987) also provides guidance on developing project plans.

Deliverables:

Develop a Project Plan that provides detail for the Planning and Requirements Definition Phases and high-level information for the other lifecycle phases. At the conclusion of each phase, the Project Plan will be reviewed to determine if the project estimates for resources, cost, and schedule need to be revised for either the current phase or subsequent phases. The software planning data shall be managed and controlled. In addition, the Project Plan will be expanded to provide detailed estimates of resources, costs, and hours for the next phase.

Some of the critical elements of a Project Plan include:

- Project Overview including list of deliverables
- Project Organization including organizational structure, project responsibilities, and process model
- Management Process including assumptions, dependencies, and constraints; risk management; tracking and oversight, and staff planning
- Technical Approach including development lifecycle, other methods, tools, techniques, project documentation, and support functions
- Work Packages, Schedule and Budget including resource requirements, budget allocations, schedule, and work breakdown structures

Review Process:

Conduct a structured walkthrough to ensure that the Project Plan reflects the project objectives and scope; identifies and mitigates project risks, and adequately estimates the project resources, costs, and schedule. The software project manager and other affected groups should review the Project Plan.

Tasks:

The following tasks are involved in developing the project plan:

Develop Software Quality Assurance Plan

Responsibility: Project Manager and Quality Assurance Manager

Description:

The software quality assurance program involves the reviewing and auditing of the software products and activities to verify that they comply with the applicable
procedures and standards and to assure the production and operation of high quality products according to stated requirements. The results of these reviews and audits provide the project manager and other appropriate managers with appropriate visibility into the processes used.

The software quality assurance program is initiated at the beginning of a project and is conducted throughout the systems development lifecycle. The software quality assurance program is the joint responsibility of the project manager and quality assurance manager with direct support and involvement from the quality assurance representatives assigned to the project. Software Quality Assurance includes the following activities:

- Preparation of a Software Quality Assurance Plan for the project according to a documented procedure and coordinated with the designated project team members.
- Establishment of a reporting channel regarding software quality assurance.
- Allocation of adequate resources and funding to maintain and perform software quality assurance activities.
- Provision of formal training for the software quality assurance representatives to perform their activities.
- Provision of orientation training in the role, responsibilities, authority and value of software quality assurance to project software developers.
- Performance of software quality assurance activities in accordance with the Software Quality Assurance Plan.
- Participation by software quality assurance representatives in the preparation and review of the Projects Plan, standards, and procedures.
- Reviews by the software quality assurance representatives of the systems development activities to verify compliance.
- Audits by software quality assurance representatives of designated deliverables to verify compliance.
- Periodic reports by software quality assurance representatives of the results and deliverables of software quality assurance activities to the project team.
- Documentation of deviations identified in the project activities and

deliverables and handled in accordance with a documented procedure.

- Use of measurement and analysis techniques to determine the cost and schedule status of the software quality assurance activities.
- Periodic reviews by independent software quality assurance representatives of the activities and software deliverables of the projects software quality assurance program.

Software quality assurance personnel work with the project manager during early phases to establish plans, standards, and procedures that will add value to the
software product and satisfy the constraints of the project and the organization's policies. By participating in establishing software quality deliverables, e.g., the plans, standards, and procedures, the software quality assurance personnel help ensure the deliverables fit the projects needs and will be usable for performing reviews and audits throughout the project lifecycle.

Compliance issues are first addressed with the project manager and resolved there if possible. For issues not resolvable by the software quality assurance representative and project manager, the issue is escalated to an appropriate level of management for resolution .

Deliverables:

The quality assurance manager or designated representative assists the project manager with the development of a plan that clearly defines the project's quality assurance policies and procedures. The Software Quality Assurance Plan addresses the following types of activities:

- Establishing the applicability of published standards and procedures and determining the scope of the project standards and procedures.
- Monitoring the software product and enforcement of compliance with all standards and procedures to facilitate the early detection of problems that could affect the reliability, maintainability, availability, integrity, safety, security, or usability of the software product.
- Inspecting hardware and software items and documenting for compliance to specifications and standards before their release to the test team or the system owner.
- Certifying deliverable items before their release to the system owner as compliant with all provisions of the project statement of work and contract, if applicable.
- Coordinating the project's technical problem reporting system and corrective action program to assure resolution of observed discrepancies.
- Measuring the quantitative and auditable progress of the project based on cost, schedule status, and quality status.
- Assuring consistent management and technical practices and the

Develop Software Quality Assurance Plan

integrity of the software product.

Provide enough information in the plan so that compliance can be monitored by means of project records. Whenever feasible, acquire automated tools to check compliance with project standards. For example, many CASE (computer-aided software engineering) tools can check compliance with standards, while checking the validity and consistency of requirements, design, and logic diagrams.

Conduct a structured walkthrough to validate that the quality assurance policies and procedures are appropriate and adequate for the project.

Review Process:

The following resources are available to assist with the development of a Software Quality Assurance Plan on Web site at: http://www.michigan.gov/dit

- Software Quality Assurance Plan Template
- Software Quality Assurance Plan Example

Resources:

The Institute of Electrical and Electronics Engineers (IEEE)Standard for Software Quality Assurance Plans (Std 730-1989) provides guidance on developing these plans.

Develop Software Configuration Management Plan

Responsibility:

Project Manager or Software Configuration Manager

Description:

Software configuration management (SCM) is a set of procedures used to control changes to the project during all phases of the software lifecycle. The goals of configuration management is to identify the configuration of the software (i.e., software deliverables and their descriptions) at given points in time, to systematically control changes to the configuration, and to maintain the integrity and traceability of the configuration throughout the software lifecycle.

Software configuration management includes the following activities:

- A Software Configuration Management Plan is prepared for each software project according to a documented procedure and is coordinated with affected groups and individuals.
- A documented and approved software configuration management plan is used as the basis for performing the software configuration management activities.
- Authority for managing the project's software baselines is established (e.g. Software Configuration Control Board - SCCB).
- A configuration management library system is established as a repository for the software baselines.
- Software deliverables are identified and placed under configuration control management.
- Change requests and problem reports for all software items/units are initiated, recorded, reviewed, approved, and tracked according to a documented procedure.
- Changes to baselines are controlled according to a documented procedure.
- Products from the software baseline library are created and their release is controlled according to a documented procedure.
- Status of software items/units is recorded according to a documented procedure.
- Standard reports documenting the software configuration management activities and the contents of the software baseline are developed and made available to affected groups and individuals.
- Software baseline audits are conducted according to a documented procedure.

Deliverables:

The deliverables placed under software configuration management include the software products that are delivered to the customer (e.g., the software requirements document and the source code) and the items that are identified with or required to create these software products (e.g., the compiler). A software baseline library is established containing the software baselines of the configuration items as they are developed. Changes to baselines and the release of software products built from the software baseline library are systematically controlled via the change control and configuration auditing functions of software configuration management.

The Software Configuration Manager (or the individual assigned software configuration responsibilities) is responsible for routine evaluation of the software product. The Software Configuration Manager controls changes that are introduced into the software product environment. The SCM

Develop Software Configuration Management Plan

manager is responsible for the processes necessary to correct faults in the environment and
software product. The SCM manager is not responsible for any overall project deadlines or management issues.

Deliverables Continued:

A Software Configuration Management Plan that defines the configuration management policies and procedures is required for each software project. The plan is developed early in the lifecycle to ensure the control of changes as soon as the project requirements are approved and baselined. In this phase, the plan addresses activities that are platform independent, such as identifying the items that will be placed under configuration management. As the project progresses through the lifecycle phases, the plan is expanded to reflect platform specific activities.

The Software Configuration Management Plan addresses the following types of responsibilities and activities:

- Defining the required software configuration management policy and procedures.
- Maintaining all documents in a central library where they can be accessed with ease.
- Receiving unit test and integrated software builds and related documentation from the developer.
- Performing version control procedures on unit and integrated software builds received from the developer.
- Packaging tested unit or integrated build for return to developer or production as required.
- Shipping approved unit or integrated build to production as required.
- Maintaining an archive of project related correspondence between members of the group.
- Overseeing the release and subsequent distribution of configuration items.

Provide enough information in the plan so that compliance can be monitored by means of project records. Whenever feasible, acquire automated Software Configuration Management tools to check compliance with project standards, the validity and consistency of software product design, requirements and system performance.

Based on the complexity of the project and the anticipated volume of changes, a Software Configuration Management Plan can be developed for a specific project, an existing plan can be modified to suit the requirements of a project, or a plan can be developed to manage all of the projects supporting a particular system owner's organization. Place a copy of the Software Configuration Management Plan in the Project Notebook.

Develop Software Configuration Management Plan

Review Process:	Conduct structured walkthroughs to validate that the configuration management approach, the configuration identification, change control, status accounting, and auditing procedures are appropriate for the project.
Resources:	Software Configuration Management Plans and a template are available on the Research and Policy Web site at: http://www.michigan.gov/dit . The Institute of Electrical and Electronics Engineers (IEEE)Standard for Software Configuration Management Plans (Std 828-1990) provides guidance on developing these plans.

Investigate Software Alternatives

Description:

1) *Investigate software availability;*
2) *Investigate software alternatives;*
3) *Build versus Buy*

When the software to be used for the project has not been predetermined by the system owner's existing computing environment, software available within the Agency and the commercial marketplace should be investigated. In the Planning Phase, the investigation of software alternatives is geared to determining project feasibility.

Unless the cost effectiveness of developing custom-built software to meet mission needs is clear and documented, all sources of reusable code, applications, and commercial-off-the-shelf (COTS) software must be investigated on a site and Agency-wide basis prior to making a decision to custom-build code for the project. This practice ensures the most cost-effective and efficient use of resources, and will decrease the number of duplicative and overlapping software systems. The choice to develop a customized application should be balanced against the availability of other solutions; and the project cost, resources, and time constraints.

Software Alternatives:

The following is a list of software alternatives that should be considered:

- ❑ Adapt existing software in use within the agency.
- ❑ Adapt existing software in use within other government agencies.
- ❑ Adapt mainframe or minicomputer source code obtained from Agency repositories.
- ❑ Purchase commercial-off-the-shelf (COTS) software.
- ❑ Reuse existing modules of code.
- ❑ Adapt reusable code to fit the new application.
- ❑ Develop a custom-built software product.

Note: Medium and small software development efforts are often restricted to the system owner's existing software. This should not preclude the potential cost savings of reengineering existing software modules rather than custom building the entire software system.

Investigate Hardware Alternatives

Description:

1) Investigate hardware alternatives

When the hardware to be used for the project has not been predetermined by the system owner's existing computing environment, investigate hardware available and through the commercial marketplace. In the Planning Phase, the investigation of hardware is geared to determining project feasibility.

Factors to Consider:

The following is a list of factors that should be considered when identifying hardware alternatives:

- ❑ Availability and cost of hardware
  - Shareable hardware
  - Government excess
  - New procurement
- ❑ Architecture compliance
- ❑ Current and future communications needs
- ❑ Computer security requirements of the system
- ❑ Standards requirements
- ❑ Volume of data
- ❑ Importance of data to the Agency mission
- ❑ Importance of data to the customer organization's mission and to job performance
- ❑ Potential growth of the software to serve more customers
- ❑ Potential growth of the software to serve more locations
- ❑ Potential for interface to other systems or organizations
- ❑ Conformance to government standards such as networking and open systems

***Note**:* Medium and small software development efforts are often restricted to the system owner's or customer sites' existing hardware.

Formulate Platform Options

Description:

1) *Formulate preliminary platform options;*
2) *Identify the:*
   - *Benefits*
   - *Costs*
   - *Assumptions*
   - *Constraints*
   - *Dependencies*
   - *Risks*

Use the information collected about software and hardware alternatives to formulate preliminary platform options. The purpose of identifying platform options early in the project lifecycle is to assure that at least one technically feasible and cost-effective approach exists to satisfy the project objectives. If more than one platform option is feasible, identify the benefits, costs, assumptions, constraints, dependencies, and risks associated with each option.

No platform decisions are made at this time. Detailed technical solutions are premature prior to defining the product requirements. The platform alternatives information gathered in the Planning Phase is revisited in the Functional Design Phase, at which time a final recommendation is developed by the project team and presented to the system owner. The system owner is responsible for making the final platform decision.

Deliverables:

Develop a summary of platform options for use in the Feasibility Review or Project Feasibility . Place a copy of the platform option information in the Project Notebook.

Review Process:

Conduct a structured walkthrough to ensure that the most viable platform options have been identified.

Conduct Project Reviews

Conduct Structured Walkthroughs

Responsibility:	Project Manager, Work Product Author and Reviewers
Description:	During the Planning Phase, schedule at least one structured walkthrough to review each of the Planning Phase deliverables, i.e., project plan, feasibility study, software quality assurance plan, configuration management plan, etc. ***Note:*** A structured walkthrough is an effective project management tool that is recommended for all projects regardless of size; however, if an item has gone through a prior formal structured walkthrough, and the modifications are minor, a less formal review may be warranted. *Appendix C, Conducting Structured Walkthroughs*, provides a procedure and sample forms that can be used for structured walkthroughs.
SDLC Reference:	

Conduct In-Phase Assessment

Responsibility:	Project Manager and Independent Reviewer
Description:	Schedule at least one IPA prior to the Planning Phase Exit process. Additional IPAs can be performed during the phase, as appropriate. An IPA is recommended after the completion of the Planning documents. Specification. ***Note:*** An IPA is an effective project management tool that is recommended for all projects regardless of size.
SDLC Reference:	*Appendix D, In-Phase Assessment Process Guide*, provides a procedure and sample report form that can be used for In-Phase Assessments.

Conduct Requirements Definition Phase Exit

Responsibility:	Project Manager
Description:	Schedule the Phase Exit as the last activity of the Planning Phase. It is the responsibility of the project manager to notify the appropriate participants when a project is ready for the Phase Exit process and to schedule the Phase Exit meeting. All functional areas and the Quality Assurance representative involved with the project should receive copies of the deliverables produced in this phase. ***Note:*** A Phase Exit is an effective project management tool that is recommended for all software projects regardless of size. For small software projects, phases can be combined and addressed during one Phase Exit.
SDLC Reference:	*Appendix E, Phase Exit Process Guide*, provides a procedure and sample report form that can be used for phase exits.

Application Security Diagnostic Tool

Prupose:

This *e-Government Application Security Diagnostic Tool* is designed to assist the design professional with security risk assessment and works in conjunction with the *Security Risk Assessment Guidelines* document. This tool presents issues relevant to application security and risk in an interactive format taking into account security risk and the severity level of a breach.

Webster's definition of risk:

> **Risk** is the possibility of suffering loss.

In a development project, the loss describes the possibility of losing the integrity of data and information.

Each risk nominally goes through these functions sequentially, but the activity occurs continuously, concurrently (e.g., risks are tracked in parallel while new risks are identified and analyzed), and iteratively (e.g., the mitigation plan for one risk may yield another risk) throughout the project life cycle.

Function	Description
Identify	Search for and locate risks before they become problems.
Analyze	Transform risk data into decision-making information. Evaluate impact, probability, and timeframe, classify risks, and prioritize risks.
Plan	Translate risk information into decisions and mitigating actions (both present and future) and implement those actions.
Track	Monitor risk indicators and mitigation actions.
Control	Correct for deviations from the risk mitigation plans.
Communicate	Provide information and feedback internal and external to the project on the risk activities, current risks, and emerging risks. *Note*: Communication happens throughout all the functions of risk management.

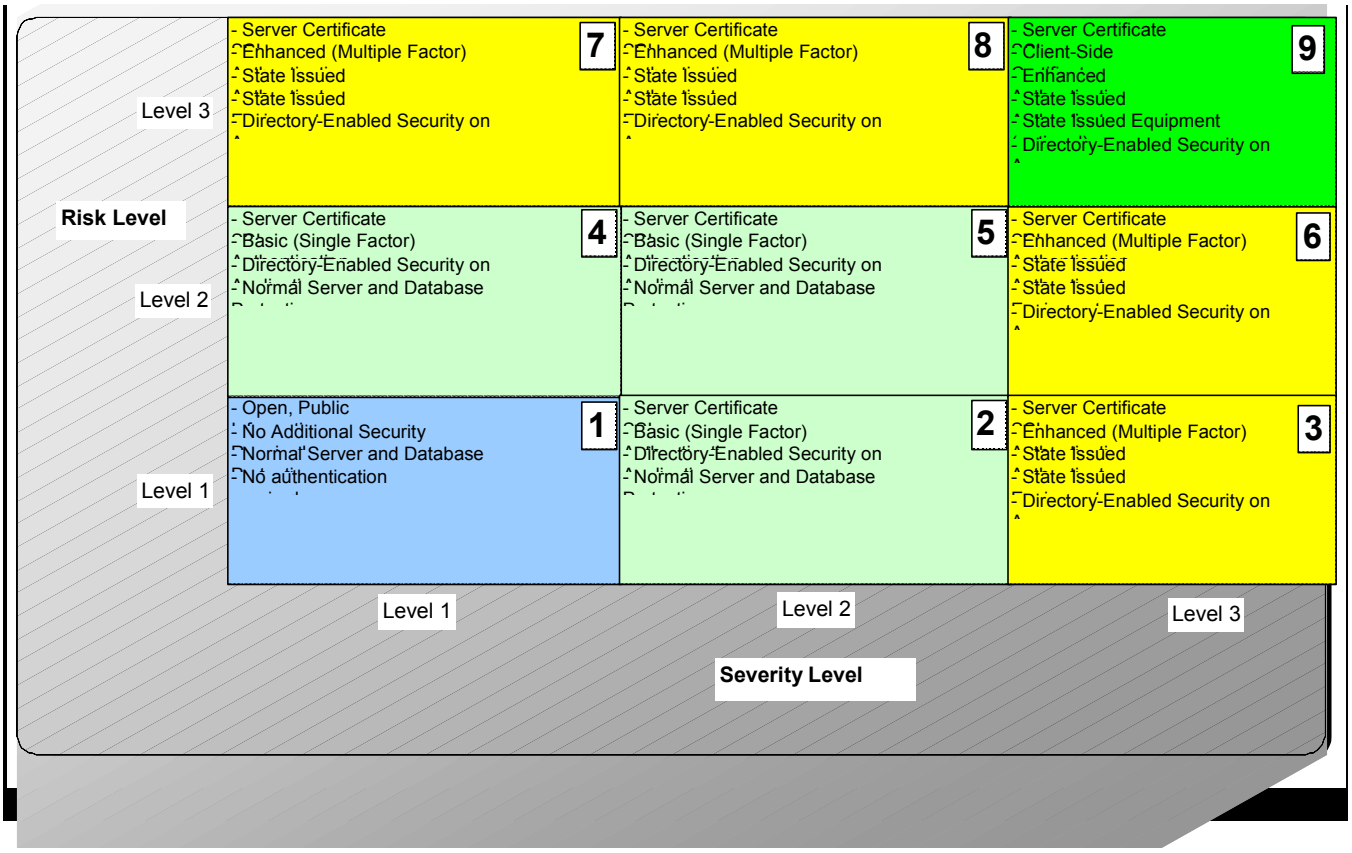
Application Security Diagnostic Tool

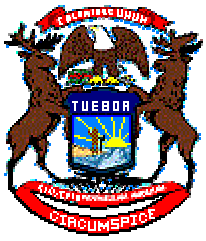
Scope:

This tool is focused on application risk assessment and does not attempt to address security issues associated with technical hardware or software. Application risk assessment is important during the systems development lifecycle and should be factored in as an important component of this processes. In addition, existing applications can be assessed with this tool.

The e-government application security diagnostic tool is available on the Research and policy Web site at: http://www.michigan.gov/dit . The enterprise IT standard for this tool is **1350.50 - Use of PKI Certificates and Web Browser Application Risk Assessment** is available by request from the Office of Research and Policy.

Security Risk/Severity Level Graph





SYSTEMS DEVELOPMENT LIFECYCLE

SECTION 5

REQUIREMENTS DEFINITION PHASE

Section 5: Requirements Definition Phase

Table of Contents

Section 5: Requirements Definition Phase

Highlights of Phase

Requirements Definition Phase

Inputs:

- Project Notebook
- Description of customer environment
- Statement of project scope and objectives
- Statement of high-level project requirements
- Functional area contact list and project profile
- Summary of platform options
- Statement of project feasibility
- Cost Benefit Analysis
- Feasibility Study Document
- Project Plan
- Software Quality Assurance Plan
- Software Configuration Management Plan
- Risk Plan
- Communications Plan

Outputs:

- Description of analysis technique
- Records of all project requirements
- Customer-oriented requirements manual *(optional)*
- Continuity of Operations Statement/Plan
- Data Dictionary
- Requirements Traceability Matrix
- Software Requirements Specification
- Project Test Plan
- Acceptance Test Plan *(draft)*
- Design technique
- Project Plan *(revised)*

Key Activities:

Requirements Management
Select Requirements Analysis Technique
Define Project Requirements
Compile and Document Project Requirements
Establish Functional Baseline
Develop Project Test Plan
Develop Acceptance Test Plan
Select Design Technique
Revise Project Plan
Conduct Structured Walkthroughs
Conduct In-Phase Assessment
Conduct Requirements Definition Phase Exit

Methods and Tools

Structured Walkthroughs

Peer Reviews

In-Phase Assessment

Quality Reviews

Phase Exit

Forms Utilized

Requirements Traceability Matrix
Software Requirements Specifications
Software Change Control Request Template
Software Change Control Log
Continuity Operations Plan
Project Test Plans, Acceptance Test Plan

Overview

Description:

1) *Develop a basis of mutual understanding of project requirements;*
2) *Obtain an approved software requirements specification;*
3) *Analyze customers business processes and needs;*
4) *Develop formal requirements document;*
5) *Plan test activities*

The primary goal of the requirement definition phase is to develop a basis of mutual understanding between the system owner/customers and the project team about the requirements for the project. The result of this understanding is an approved Software Requirements Specification that becomes the initial baseline for software product design and a reference for determining whether the completed software product performs as the system owner requested and expected.

This phase involves analysis of the system owner/customers' business processes and needs, translation of those processes and needs into formal requirements, and planning the testing activities to validate the performance of the software product.

Review Process:

Quality reviews are necessary during this phase to validate the product and associated deliverables. The activities that are appropriate for quality reviews are identified in this section and the *Section 2 Lifecycle Model.* The time and resources needed to conduct the quality reviews should be reflected in the project resources, schedule, and work breakdown structure.

SDLC References:

Section 2 Lifecycle Model,Quality Reviews, provides an overview of the Quality Reviews to be conducted on a project.

Appendix C, Conducting Structured Walkthroughs, provides a procedure and sample forms that can be used for structured walkthroughs.

Appendix D, In-Phase Assessment Process Guide, provides a procedure and sample report form that can be used for in-phase assessments.

Appendix E, Phase Exit Process Guide, provides a procedure and sample report form that can be used for phase exits.

Requirements Management

Responsibility:

Project Manager

Description:

Requirements management is essentially a program composed of gathering, organizing, prioritizing, and documenting requirements; verifying that requirements have been captured in the product, and managing changes to requirements. Gathering, organizing, prioritizing and documenting requirements is an interactive communication process and working relationship between stakeholders and the project team to discover, define, refine, and record a precise representation of the product requirements.

1) *Gathering;*
2) *Organizing;*
3) *Prioritizing;*
4) *Documents;*
5) *Verifying; and*
6) *Managing requirements*

Requirements management documents the needs, expectations, and understanding of the product to be delivered and provides a framework for identifying, planning, scheduling, costing, verifying, tracing, testing, evaluating, changing, and renegotiating requirements to satisfy stakeholder needs and expectations of the project. When requirements are initially gathered, some or all will be planned for the current project (e.g., initial release). The requirements for the project are documented in the Software Requirements Specification. As the project progresses, more requirements may be identified and managed through a change control process. As part of requirements management, the project manager must track requirements that are accepted for the current project and those which will be planned for subsequent releases.

Each requirement identified in the Software Requirements Specification document should be uniquely identified in a Requirements Traceability Matrix. The Requirements Traceability Matrix is a requirement management tool that ensures requirements are traced and verified through the various lifecycle phases, especially design, testing, and implementation. Requirements must be traceable from external sources such as the customer, to derived system-level requirements, to specific hardware/software product requirements. In addition, all of these requirements must be cross-traceable to design, implementation, and test artifacts to ensure requirements have been satisfied.

Deliverables:

A substantial amount of information that is used for requirements management in later phases in the software development process is gathered in the Requirements Definition Phase. The Requirements Traceability Matrix is a deliverable that is created during the Requirements Definition Phase and used to verify and validate that requirements are met and the product remains within scope. Refer to each task for information on applicable deliverables.

Review Process:

Refer to each task for applicable review processes.

Resources:

A template for the Requirements Traceability Matrix and Software Requirements Specification document are available on the Research and Policy Web site http://www.michigan.gov/dit

Carnegie Mellon University, Software Engineering Institute, *Capability Maturity Model: Guidelines for Improving the Software Process*, Addison Wesley Longman, Inc., 1994: http://www.sei.cmu.edu

Tasks:

The following tasks are involved in requirements management:

Develop Responsibility Traceability Matrix

Requirements Management

Requirements Change Control

Requirements Traceability Matrix

Description:

A requirements traceability matrix is a table used to trace project lifecycle activities and deliverables to the project requirements. The matrix establishes a thread that traces requirements from identification through implementation.

Every project requirement must be traceable back to a specific project objective(s) described in the Project Plan. This traceability assures that the product will meet all of the project objectives and will not include inappropriate or extraneous functionality.

All deliverables produced during the design, code, and testing processes in subsequent lifecycle phases must be traced back to the project requirements described in the Software Requirements Specification. This traceability assures that the product will satisfy all of the requirements and remain within the project scope.

It is also important to know the source of each requirement, so that the requirements can be verified as necessary, accurate, and complete. Meeting conference records, customer survey responses, and business documents are typical sources for project requirements.

Deliverables:

Develop a matrix to trace the requirements back to the project objectives identified in the Project Plan and forward through the remainder of the project lifecycle phases. Place a copy of the matrix in the Project Notebook. Expand the matrix in each phase to show traceability of deliverables to the requirements and vice versa.

The requirements traceability matrix should contain the following fields:

- ❑ The requirement statement.
- ❑ Requirement source (Conference; Configuration Control Board; Task Assignment, etc.).
- ❑ Software Requirements Specification and/or Functional Requirements
- ❑ Document paragraph number containing the requirement.
- ❑ Design Specification paragraph number containing the requirement.
- ❑ Program Module containing the requirement.
- ❑ Test Specification containing the requirement test.
- ❑ Test Case number(s) where requirement is to be tested (optional).
- ❑ Verification of successful testing of requirements.
- ❑ Modification field. If requirement was changed, eliminated, or replaced, indicate disposition and authority for modification.
- ❑ Remarks.

Requirements Traceability Matrix

Review Process:

Conduct a structured walkthrough of the Requirements Traceability Matrix to ensure that all requirements have been accurately captured.

Illustration of Traceability Matrix:

One method for tracing requirements is a threading matrix that groups requirements by project objectives. Under each project objective, the source of the Requirement, the unique requirement identification number, and the lifecycle activities are listed in columns along the top and the project requirements in rows along the left side. As the project progresses through the lifecycle phases, a reference to each requirement is entered in the cell corresponding to the appropriate lifecycle activity. *See Sample Requirements Traceability Matrix below,* provides an example.

Resource:

A template for the Requirements Traceability Matrix is available on the Web site at: http://www.michigan.gov/dit .

Requirement	Source	Unique Number	Software Reqts Spec/Functional Requirement Document	Design Spec.	Program Module	Test Spec	Test Case(s)	Successful Test Verification	Modification of Reqt.	Remarks
Objective 1: Security										
The software product shall have three customer access levels with the capability to add new access levels in the future.	9/10/01 security meeting									
Each customer access level shall have a unique designation.	9/10/01 security meeting									
One customer access level shall allow read-only access to the production database.	9/10/01 security meeting									

Requirements Change Control

Description:

As a project progresses, more requirements may be identified. Using the change control process, the project manager will track requirements that are accepted for the current project and those which will be planned for subsequent releases.

Changes to requirements should be initiated via a formal change request form, and then logged and tracked by the project manager to ensure the changes are included in the traceability matrix, testing and acceptance plans. Other deliverables are revised as appropriate.

When the project manager is alerted to a change in requirements, the person initiating the change should document the request on a software change request form. The change request form should capture as much detail about the requirement as possible, e.g.; its effect on the system, procedures and documentation; as well as the reason and priority of the change. A separate change request form should be completed for each requested change.

When the project manager receives the formal software change request form, the change should be recorded on a software change control log. Once logged, the request should be provided to the development staff for evaluation.

Once the evaluation is completed, the project stakeholders and project manager should evaluate the impact to the project and the priority level of the change request to determine whether or not to approve the change for inclusion in the current project or deferral for a future project. Approvals should be recorded on the change request form.

The status of the change request should be managed through on the software change control log and updated as the status changes.

Once a change is approved, the requirements traceability matrix and all other appropriate deliverables, e.g., test plans or acceptance plans, should be updated to include the new requirement. If scheduling is impacted by the change, the Project Plan should be updated.

Deliverables:

As changes to the requirements are requested, the completed Software Change Request Forms and a copy of the Software Change Control Log should be maintained in the Project Notebook.

Review Process:

A peer review or structured walkthrough may be conducted on the Software Change Request Forms and Software Change Control Log.

Resources:

Templates for the Software Change Control Request and the Software Change Control Log can be found on the DIT web http://www.michigan.gov/dit .

Select Requirements Analysis Techniques

Responsibility:

Project Manager/Team

Description:

A requirements analysis technique is the set of data collection and analysis techniques (e.g., customer interviews and rapid prototyping) combined with the lifecycle requirements standards (e.g., tracing the requirements through all lifecycle activities) that are used to identify the project requirements and to define exactly what the software product must do to meet the system owner/customers' needs and expectations. When appropriate, the technique must include methods for collecting data about customers at more than one geographic location and with different levels and types of needs.

The requirements analysis technique should be in harmony with the type, size, and scope of the project; the number, location, and technical expertise of the customers; and the anticipated level of involvement of the customers in the data collection and analysis processes. The technique should ensure that the functionality, performance expectations, and constraints of the project are accurately identified from the system owner/customers' perspective. The technique should facilitate the analysis of requirements for their potential impact on existing operations and business practices, future maintenance activities, and the ability to support the system owner's long-range information resource management plans.

It is advantageous to select a technique that can be repeated for similar projects. This allows the project team and the system owner/customers to become familiar and comfortable with the technique.

Discuss the analysis technique with the system owner and customers to make sure they understand the process being used, their role and responsibilities in the process, and the expected format of the output (e.g., how the requirements will be organized and described).

Deliverables:

Create a description of the analysis technique and share it with all members of the project team, system owner, and customers. Place a copy of the analysis technique description in the Project Notebook.

Review Process:

Conduct a structured walkthrough to verify that the requirements analysis technique is appropriate for the scope and objectives of the project. A structured walkthrough is not needed when the technique has been used successfully on similar projects for the same system owner/customer environment.

Define Project Requirements

Responsibility: Project Manager/Team

Description:

Use the project scope, objectives, and high-level requirements as the basis for defining the project requirements. The questions used to define the project objectives may be helpful in developing the project requirements. The goals for defining project requirements are to identify what functions are to be performed on what data, to produce what results, at what location, and for whom.

The requirements must focus on the software products that are needed and the functions that are to be performed. Avoid incorporating design issues and specifications in the requirements. One of the most difficult tasks is to determine the difference between "what" is required and "how to" accomplish what is required. Generally, a requirement specifies an externally visible function or attribute of a system—i.e., "what". A design describes a particular instance of how that visible function or attribute can be achieved—i.e., "how to".

Requirements should be specified as completely and thoroughly as possible. The requirements must support the system owner's business needs, information resource management long-range plans, and the organizational and agency missions. When requirements are being defined, it is not sufficient to state only the requirements for the problems that will be solved; all of the requirements for the project must be captured.

Characteristics:

Each requirement must be stated as a unique objective with the following characteristics. The existence of these characteristics must be verified prior to the delivery of the Software Requirements Specification later in the Requirements Definition Phase:

- **Necessary** - Absolute requirements that are to be verified are indicated by "must" or "shall". Goals or intended functionality are indicated by "will".
- **Correct** – Each requirement is an accurate description of a feature or process of the software product.
- **Unambiguous** - The statement of each requirement denotes only one interpretation.
- **Complete -** Each requirement describes one result that must be achieved by the software product. The requirement should not describe the means of obtaining the result.
- **Consistent** - Individual requirements are not in conflict with other requirements.
- **Verifiable** (testable) - Each requirement is stated in concrete terms and measurable quantities. A process should exist to validate that the software product (when developed) will satisfy the set of requirements.

Characteristics Continued:

- **Modifiable** - The structure and style of the requirements are such that any necessary changes to the requirements can be made easily, completely, and consistently.
- **Traceable** - The origin of each requirement is clear and can be tracked in future development activities and tests.

Identification System:

The creation of a standard identification system for all requirements is required in order to facilitate configuration control, requirements traceability, and testing activities. The identification system must provide a unique designator for each requirement. For example, the identification system can classify the requirements by type (e.g., functional, input, or computer security). Within each type classification, the requirements can be assigned a sequential number. Select an identification system that is appropriate for the scope of the project.

Changes:

As the project evolves, the requirements may change or expand to reflect modifications in the customers' business plans, design considerations and constraints, advances in technology, and increased insight into customer business processes. A formal change control process must be used to identify, control, track, and report proposed and approved changes. Approved changes in the requirements must be incorporated into the Software Requirements Specification in such a way as to provide an accurate and complete audit trail of the changes. This change control process should be an integral part of the project's Software Configuration Management Plan.

Description of Customer Environment:

1) *Understand current customers' environment;*
2) *Analyze customers' manual procedures or automated processes;*
3) *Gain understanding of the functions performed*

A thorough understanding of the current customers' environment is necessary to define the objectives, scope, and high-level requirements of the project. Analyze the procedures manual or automated processes to understand what customers do, how they do it, and what improvements are desired or needed. This includes gaining an understanding of the functions performed, identifying information flows within the processes, and listing process inputs and outputs.

Use appropriate data collection techniques such as customer surveys, interviews, and document inspections to gather data and analyze the customer environment.

Types of Information:

The following list provides samples of the type of information that should be considered:

- **Mission** – Obtain a copy of the mission of the primary customer organization(s) and place in the project notebook.
- **Work Processes** - the customers perform Analyze the work processes or tasks that. Identify the relationships and priority of the processes.
- **Workload** - Describe the volume of work currently being performed. For automated processes include processing time for batch operations, response times, peak number of simultaneous customers of interactive systems, and number of transactions.

Define Project Requirements

Types of Information Continued:

- **Processing/Data Flow** - Analyze the major processing/data flows for the work processes. Include the flow of data between different customer groups, manual and automated processes, and different customer sites.
- **Integration/Interfaces** - Identify interactions and interfaces that the customers' current automated systems share with other automated systems.
- **Customers** - Identify the skill sets in order to assess the capability of the organization and number of personnel at both the owner's site and other SOM sites for contractor staff who operate, maintain, and use the current manual procedures or automated processes.
- **Costs** - Itemize costs incurred in operating the customers' current manual or automated systems.
- **Equipment** - Identify equipment used in the current manual or automated systems and relates equipment to the function it supports in the systems.
- **Communications** - Identify the guidelines, standards, equipment and software to support system communications.
- **Software** - Identify software packages that are being used.
- **Statutory Requirements** - Identify the guidelines, directives and standards the customer must comply with in the performance of the work processes.

Deliverables of Customer Environment:

Develop a description of the customer environment and place a copy in the Project Notebook. The description will be incorporated into future deliverables such as the Project Plan and the Requirements Specifications.

Review Process:

A peer review or structured walkthrough(s) may be conducted on the customer environment description to ensure all pertinent information has been captured.

Description of High-Level Project Requirements:

1) *Feasibility of project;*
2) *Estimate resources for hardware and software;*
3) *Estimate need for equipment or software training*

High-level requirements should be of sufficient detail to make a preliminary determination about the feasibility of the project, to estimate the resources that are needed, to assess hardware and software requirements, and to estimate the need for equipment or software training.

The current and anticipated needs of all customer groups must be identified. Customers in different organizational units or geographic locations may have diverse or unique requirements that must be incorporated into the project requirements.

The project team participates by providing technical assistance, i.e., to determine the validity of high-level customer/client information system requirements and those that can be accomplished. The group also provides available input by identifying concerns (implicit requirements) that will surface when customer requirements are implemented, i.e., hardware, software, costs (indirect/direct), people resources, training, etc.

Illustration of High-Level Project Requirements:

Organize high-level project requirements into categories of related data. The following list provides samples of the types of data that should be considered:

- **Inputs** - Identify source documents and data that will be used as input to the processes. Provide descriptive information about data such as the type, volume, condition (e.g., edited or unedited), organization, and frequency. Include inputs such as records or batch files from other systems that will be downloaded or migrated.
- **Outputs** - Identify outputs such as reports, display screens, documents, and data files.
- **Databases** - Estimate the high-level contents, purpose, use, format, organization, and update frequency of databases that will be used by the product. Identify other existing or planned databases that would interface with the product as a provider or recipient of information.
- **Processing/Data Flow** - Describe the major processing/data flow for the product. Include flow of data from the product to other systems and vice versa.
- **Data Communications** - Estimate the major data communications resources required to support the product. Include requirements for networks, dial-up access, and other communication configurations to support data access and retrieval requirements.
- **Interfaces** - Identify any systems with which the product must interface. Describe factors that may impact the design of the product.
- **Security, Privacy, and Control** - State requirements for ensuring the integrity of the data, for safeguarding against unauthorized access to the databases, and for other customer access controls.
- **Standards and Guidance** - Describe the system or process actions or data attributes that are needed to comply with standards and guidance.
- **Training** - Identify the type of training required to ensure efficient operation of the software product. Provide estimates of the number of personnel to be trained by type and frequency of training.
- **Workload** - Estimate the volume of work to be handled at slow, normal, and peak periods. Identify dates associated with each period. Include processing time for batch systems, response times, peak number of simultaneous customers of interactive systems, and number of transactions.
- **Costs** - Estimate initial development costs and expected operating cost savings over the expected lifetime of the software product.
- **Equipment** - Estimate new equipment that might need to be acquired or manufactured and current equipment that would continue to be used. Determine critical computer resources.

Illustration of High-Level Project Requirements Continued:

- **Software** - Estimate software and firmware packages that might need to be acquired and any updates needed for existing software. Determine critical computer resources.
- **Documentation -** Determine documentation needs based on *Exhibit 2.0-1*, agency-specific requirements and deliverables and other needs unique to the project.
- **Usability –** Determine the ease with which a customer can learn to operate, prepare inputs for, and interpret outputs of a system or component.
- **Operability** - Define requirements associated with the operation of the system or application. The objective is to develop an automated system that requires minimal operator intervention after initial setup. Requirements should address areas such as unattended operations, automated scheduling of system or application tasks, remote intervention capabilities, operational documents, special conditions under which system must operate, e.g., error handling and message handling for system failure and recovery.
- **Maintainability** - Determine the ease with which a system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. Determine the ease with which the system can be retained in, or restored to, a state in which it can perform its required functions.

Deliverables of High-Level Project Requirements:

Develop a formal statement of the high-level project requirements. This statement will be incorporated into the Project Plan. The statement of requirements should be included in the Project Feasibility Document (if prepared). The high-level requirements will serve as the foundation for the software requirements developed during the Requirements Definition Phase. Place a copy of the high-level requirements in the Project Notebook.

Illustration of High-Level Project Requirements:

The following are high-level access requirements:

- Allow any customer to access the application and enter an access request.
- Have an interface to verify and maintain customer information.
- Design system to verify customer access levels.
- Allow for electronic authorizations for request verification.
- Allow for the entry, query, and maintenance of application data based on the customer access levels.

Define Project Requirements

Illustration of High-Level Project Requirements Continued:

- ❑ Provide for the capture and tracking of request data for the following request types:
  - requesting initial computer access
  - adding access levels to an existing logon identification code
  - reinstating a suspended computer access
  - deleting an existing computer access
  - suspending an existing computer access
- ❑ Provide for the entry, query, and maintenance of the following information:
  - computer systems
  - applications
  - customer logon identification codes
- ❑ Allow customers to view and maintain their own address information
- ❑ Provide a means for the system owner and security officers to review and change current customer access information.

Review Process of High-Level Project Requirements:

A peer review may be conducted on the formal statement of high-level project requirements although, once the Project Plan is developed, a structured walkthrough will be conducted.

SDLC Reference:

A description of the peer review process can be found in *Section 2, Quality Reviews.*

Resource

The system owner organization's information resource management long-range plan provides useful planning information for consideration when developing the requirements.

Deliverables:

For each of the following refer to each task for applicable deliverables.

Review Process:

For each of the following refer to each task for applicable review processes.

Tasks:

The following tasks are involved in developing project requirements:

Define Functional Requirements
Define Input and Output Requirements
Define Performance Requirements
Define Customer Interface Requirements
Define System Interface Requirements
Define Communication Requirements
Define Computer Security and Access Requirements
Define Backup and Recovery Requirements
Define Data Requirements
Define Implementation Requirements

Define Functional Requirements

Description:	Functional requirements define what the software product must do to support the system owner's business functions and objectives. The functional requirements should answer the following questions: ❑ How are inputs transformed into outputs? ❑ Who initiates and receives specific information? ❑ What information must be available for each function to be performed? Identify requirements for all functions whether they are to be automated or manual. Describe the automated and manual inputs, processing, outputs, and conditions for all functions. Include a description of the standard data tables and data or records that will be shared with other applications. Identify the forms, reports, source documents, and inputs/outputs that the software product will process or produce to help define the functional requirements. A functional model should be developed to depict each process that needs to be Included. The goal of the functional model is to represent a complete top-down picture of the business process. Flow diagrams should be used to provide a hierarchical and sequential view of the process owner's business functions and the flow of information through the processes.
Deliverables:	Maintain a record of all functional requirements. Save for incorporation into the Software Requirements Specification. Place a copy of the functional requirements in the Project Notebook.
Optional Deliverables:	Consider developing an optional deliverable that defines how the final software product will operate to support the system owner organization's business functions and objectives. This customer-oriented requirements manual would identify processes in a narrative form from the customer's perspective and would include requirements for all functions whether they are to be automated or manual. A functional description can be developed to depict each process that will be provided. The goal is to present a complete top-down picture of the business process. This customer-oriented requirements manual can be used as an aid in validating the functional requirements and serves as the basis for the customer documentation. If a test group outside the project team is used, the test group can work with the project team to develop the manual.
Review Process:	Conduct structured walkthroughs as needed to ensure the necessity, testability, accuracy, and completeness of the functional requirements.

Illustration of Functional Requirement of the Time Clock System:

The validation rules for the time and attendance file are that the record type will contain “TA”.

Define Input and Output Requirements

Description:	Describe all manual and automated input requirements for the software product such as data entry from source documents and data extracts from other applications; include where the inputs are obtained. Describe all output requirements for the software product such as printed reports, display screens, and files; include who or what is to receive the output.
Deliverables:	Maintain a record of all input and output requirements. Save for incorporation into the Software Requirements Specification. Place a copy of the input and output requirements in the Project Notebook.
Review Process:	Conduct structured walkthroughs as needed to ensure the necessity, testability, accuracy, and completeness of the input and output requirements.

Illustration of Input Requirement of the Time Clock System:

The PP_End_Date is mandatory and must be sent in "ccyymmdd" format.

Illustration of Output Requirement of the Time Clock System:

When searching for an Department, the return display on the screen must the Department name not the 2 digit code.

Define Performance Requirements

Description:	Performance requirements define how the software product must function (e.g., hours of operation, response times, and throughput under various load conditions). The information gathered in defining the project objectives can translate into very specific performance requirements; (e.g., if work performed for an organization is mission essential to the Department, the hours of operation and throughput will be critical to meeting the mission). Also, government and SOM policy can dictate specific availability and response times.
Deliverables:	Maintain a record of all performance requirements. Save for incorporation into the Software Requirements Specification. Place a copy of the performance requirements in the Project Notebook.
Review Process:	Conduct structured walkthroughs as needed to ensure the necessity, testability, accuracy, and completeness of the performance requirements.

Illustration of Performance Requirement of the Time Clock System:

The application must be available for use from 8:00 am to 5:00 p.m. Monday through Friday.

Define Customer Interface Requirements

Description:	The customer interface requirements should describe how the customer will access and interact with the software product, and how information will flow between the customer and the software product.
Interface Issues:	A standard set of customer interface requirements may be established for the system owner organization. If not, work with the system owner and customers to develop a set of customer interface requirements that can be used for all automated products for the system owner's organization. A standard set of customer interface requirements will simplify the design and code processes, and ensure that all automated products have a similar look and feel to the customers. When other constraints (such as a required interface with another application) do not permit the use of existing customer interface standards, an attempt should be made to keep the customer interface requirements as close as possible to the existing standard.
Deliverables:	Maintain a record of all customer interface requirements. Save for incorporation into the Software Requirements Specification. Place a copy of the customer interface requirements in the Project Notebook.
Review Process:	Conduct structured walkthroughs as needed to ensure the necessity, testability, accuracy, and completeness of the customer interface requirements.

Illustration of Customer Interface Requirement:

The following are some of the issues that should be considered when trying to identify customer interface requirements:

- The customers' requirements for screen elements, navigation, and help information.
- The enterprise standards issued by the State of Michigan, and industry that apply to customer interfaces.
- The function of the customers who will access and use the product.
- The range of work that the customers will be performing with the product.

Define the customer interface requirements by identifying and understanding what is most important to the customer, not what is most convenient for the project team.

All data entry screens must include a unique screen identification number.

Define System Interface Requirements

Description:

The hardware and software interface requirements must specify hardware and software interfaces required to support the development, operation, and maintenance of the software product.

The following information should be considered when defining the hardware and software interface requirements:

- System owner and customers' computing environment.
- Existing or planned software that will provide data to or accept data from the product.
- Other organizations or customers having or needing access to the product.
- Purpose or mission of interfacing software.
- Common customers, data elements, reports, and sources for forms/events/outputs.
- Timing considerations that will influence sharing of data, direction of data exchange, and security constraints.
- Development constraints such as the operating system, data base management system, language compiler, tools, utilities, and network protocol drivers.
- Standardized system architecture defined by hardware and software configurations for the affected organizations, programmatic offices, sites, or telecommunications programs.

Deliverables:

Maintain a record of all system interface requirements. Save for incorporation into the Software Requirements Specification. Place a copy of the system interface requirements in the Project Notebook.

Review Process:

Conduct structured walkthroughs as needed to ensure the necessity, testability, accuracy, and completeness of the system interface requirements.

Illustration of System Interface Requirement:

The application must interface with the "Autotime for Windows" software and the "Tracy Time" software.

Define Communications Requirements

Description:

The communication requirements define connectivity and access requirements within and between customer locations and between other groups and applications.

The following factors should be considered when defining communication Requirements:

- Communication needs of the customer and customer organizations.
- Customer organization's existing and planned communications environment (e.g., telecommunications; LANs, WANs, and dial-up).
- Projected changes to the current communication architecture, such as the connection of additional local and remote sites.
- Limitations placed on communications by existing hardware and software including:
  - customer systems
  - applications that will interface with the product
  - organizations that will interface with the product
- Organization, government, and industry standards that define communication requirements and limitations.
- Future changes that may occur during the project.

Deliverables:

Maintain a record of all communication requirements. Save for incorporation into the Software Requirements Specification. Place a copy of the communication requirements in the Project Notebook.

Review Process:

Conduct structured walkthroughs as needed to ensure the necessity, testability accuracy, and completeness of the communications requirements.

Illustration of Communications Requirement:

The application must be able to communicate with the Lawson time record system.

Define Computer Security and Access Requirements

Description:	Develop the computer security requirements in conjunction with the system owner's Security Officer. This involvement affords early determination of classifications and levels of access protection required for the software product. If a software product under development processes sensitive personal information, appropriate safeguards must be established to protect the information from accidental disclosure. Implement applicable security procedures to assure data integrity and protection from unauthorized disclosure, particularly during development efforts. The organization that owns the data defines the data classification. The project team must be aware of all the types of data and of any classified or proprietary algorithms used in the software product.
Steps:	Use the following steps to determine computer security requirements: 1. Identify the types of data that will be processed by the software product. 2. Determine preliminary data protection requirements. 3. Coordinate with the owner of the host platform to identify existing supporting computer security controls, if applicable. 4. Incorporate security requirements into the Software Requirements Specification.
Deliverables:	Maintain a record of all security and access requirements. Save for incorporation into the Software Requirements Specification. Place a copy of the security and access requirements in the Project Notebook.
Review Process:	Conduct structured walkthroughs as needed to ensure the necessity, testability, accuracy, and completeness of the computer security and access requirements.

Illustration of Access Control Questions:

The following list provides sample questions that can be used to help define the access controls for the software product:

- What access restrictions are placed on the customers by their organization or programmatic office?
- What are the audit and other checking needs for the software product?
- What separation of duties, supervisory functions related to control, operating environment requirements, or other functions will impact the software product?
- What measures will be used to monitor and maintain the integrity of the software product and the data from the customer's viewpoint?

Illustration of Security Requirements:

A new customer ID and password will need to be established on the tandem computer system used for the electronic data exchange gateway at the Michigan Information Processing Center (MIPC).

Define Backup and Recovery Requirements

Description:	Develop the requirements for data backup, recovery, and operation startup for the software product in conjunction with the site authority for continuity of operations. If a software product has been defined as mission essential, a Continuity of Operations Plan must be developed. A checklist is provided in *Exhibit 5.3-1, Checklist for Identifying Mission-Essential Software,* to determine if the software is mission essential. Additionally, ensure that the mission essential system is included in the Continuity of Operations or Disaster Recovery Plans for the system on which the software is executed.
Deliverables:	If a software product is determined to be mission essential, Continuity of Operations Plan must be developed. If the software product is not mission essential, a continuity of operations statement is required. Two samples of continuity of operations statements that are appropriate for software that is not mission essential are provided after the checklist. Place a copy of the Continuity of Operations Statement or Plan in the Project Notebook.
Review Process:	Conduct structured walkthroughs as needed to assure the necessity, testability, accuracy, and completeness of the backup and recovery requirements.
Resource:	A template for the Continuity of Operations Plan is available on the Research and Policy Web site at: http://www.michigan.gov/dit .

Illustration of Back-up and Recovery of the Time Clock System:

The project will introduce one new daily backup processing during TKU release payroll processing days, to backup the interface file data that have been updated to the archive directory.

Exhibit 5.3-1 Checklist for Identifying Mission-Essential Software

Use this checklist to identify software products that are mission essential. IF a "yes" answer is selected for one or more of the criteria, the software product is mission essential and Continuity of Operations Plan must be developed.

	Criterion	Yes	No
1	Inability to perform function adversely affects agencies mission.		
2	Inability to perform function adversely affects safety of individuals.		
3	Needed for activities during a state emergency.		
4	Needed for mobilization and protection of material and manpower during state emergency.		
5	Function required for maintenance of public health, safety, and order.		
6	Maintains records essential to preservation of legal rights.		
7	Large financial loss incurred with inability to perform functions.		
8	Large expense incurred if performing function by other means.		
9	Primary repository of information reported to Legislature or other agencies.		
10	Critical for compliance with federal and/or state regulatory requirements.		
11	Sole source of data unobtainable by other means, or not easily recreated.		

Define Data Requirements

Description:	Data requirements identify the data elements and logical data groupings that will be stored and processed by the software product. The identification and grouping of data begins during the Requirements Definition Phase and is expanded in subsequent phases as more information about the data is known.
Deliverables:	The major output of the data requirements identification process is a data dictionary. A data dictionary provides an ordered set of definitions about data inputs and outputs, and data stores. In the Requirements Definition Phase, the data dictionary contains a minimum amount of information about data elements such as definitions of the entities, how the data are stored, and data flows to or from other applications. The data dictionary is refined during the design phases as data elements are documented in more detail, and the logical groupings of data elements are formed into interrelated tables or record descriptions. Maintain a record of all data requirements. Save for incorporation into the Software Requirements Specification. Place a copy of the data requirements in the Project Notebook.
Review Process:	Conduct structured walkthroughs as needed to ensure the necessity, testability, accuracy, and completeness of the data requirements.

Illustration of Data Requirement:

The data files sent from the state agencies should have headers and footers so that the data received is checked for any transmittal errors.

Define Implementation Requirements

Description:

Describe the requirements anticipated for implementing the software product (e.g., customer production cycle). The high-level implementation requirements are identified early in the lifecycle to support decisions that need to be made for the software development approach. The implementation requirements are expanded into a full implementation approach during the design phases. The following paragraphs provide highlights of some of the implementation requirements that need to be considered.

Operating Environment:

Identify any capacity restrictions on the existing hardware or software that needs to be addressed and identify any hardware or software that needs to be acquired (e.g., communication hardware, file servers, off-the-shelf software, network interface cards, and LAN utilities).

Acquisition: If hardware or software must be acquired, identify the necessary acquisition activities. These activities include preparing specifications, estimating costs, scheduling procurement activities, selection, installation, and testing.

Conversion: Identify requirements for converting data from an existing or external application to the new software product. Consider requirements for data entry, data protection, computer time, conversion programs, personnel, and other resources that will be needed. Also identify the requirements for the conversion of software, if necessary. Implementing a new application may involve converting software from one environment to another, or modifying software to interface with other applications. Include requirements for testing the conversion process and validating that it was successfully accomplished.

Installation: Identify the installation requirements for any new hardware, operating system, or software. For hardware installations, consider environmental factors such as air conditioning, power supply, and security requirements. For software installations, consider proprietary software such as data base management systems. For application software, consider the installation of the application's programs, parallel operation of the old and new applications, or the cutover from a test to a production environment. Hardware and software installation must be coordinated with the work cycles of the customer organization to create a minimum of disruption, and to assure that data are available as needed. Installation must be scheduled to assure that, when data conversion is necessary, the needed data are protected.

Training: Identify the specific training needs for various categories of customers and administrators. Also identify training requirements for personnel time, computer time, training facilities, and training data base(s).

Documentation: Identify requirements for the development and distribution of operational documentation for software support personnel and customer documentation.

Define Implementation Requirements

Operating Environment Continued:	Operational documentation may include job control procedures and listings, operational instructions, system administration responsibilities, archiving procedures, and error recovery. Customer documentation includes the procedures manual, step-by-step instructions, online documentation, and online help facilities.
Deliverables:	Maintain a record of all implementation requirements. Save for incorporation into the Software Requirements Specification. Place a copy of the implementation requirements in the Project Notebook. This information will also be used to develop an Implementation Plan in the Functional Design Phase.
Review Process:	Conduct structured walkthroughs as needed to ensure the necessity, testability, accuracy, and completeness of the implementation requirements.

Illustration of Conversion Requirements:

The Time Clock system will make any necessary conversions to the software.

Compile & Document Project Requirements

Responsibility:

Project Manager/Team

Description:

Compile the requirements gathered during the requirements analysis process in preparation for the development and delivery of the draft Software Requirements Specification. The following steps should be performed as part of the requirements compilation activity:

- Select and use a standard format for describing the requirements. Ensure you comply with Information Architecture standards and any site-specific standards.
- Present the logical and physical requirements without dictating a physical design or technical solutions.
- Write the requirements in non-technical language that can be fully understood by the system owner and customers.
- Organize the requirements into meaningful groupings (e.g., all security-related requirements or all requirements for generating reports).
- Develop a numbering scheme for the unique identification of each requirement.
- Select a method for: (1) tracing the requirements back to the sources of information used in deriving the requirements (e.g., specific system owner/customer project objectives); and (2) threading requirements through all subsequent lifecycle activities (e.g., testing).

Deliverables:

Refer to the task for applicable deliverables.

Review Process:

Refer to the task for applicable review processes.

Resource:

A template for a Software Requirements Specification is available on the Research and Policy Web site at: http://www.michigan.gov/dit .

Task:

The following task is involved in the compilation of the project requirements:

Develop Software Requirements Specifications

Develop Software Requirements Specification

Description:	The Software Requirements Specification describes the inputs to be supplied by the customer or other sources, the processing that needs to occur, and the outputs desired by the customer or required by interfacing systems. The emphasis should be placed on specifying product functions without implying how the product will provide those functions. This approach provides maximum flexibility for the product designers. The how-to of product implementation is determined in the design phases. Additionally, project information should not be included in a Software Requirements Specification. A project describes a particular sequence of activities and associated resources that defines a process to develop the software product. However, certain information may be considered a requirement in one project but design or implementation details in another project. The Software Requirements Specification should be carefully reviewed to ensure each documented requirement is not project or design information.
Deliverables:	Prepare the Software Requirements Specification by integrating all of the requirements developed during this phase. Several formats are available for organizing the requirements information (e.g., from a functional perspective or a data processing perspective). Document all design constraints including processing, performance, interface, resource, safety, security and reliability requirements. Define data constraints such as limits, formats, messages, commands, and displays.
Review Process:	Conduct structured walkthroughs as needed to ensure that the Software Requirements Specification is accurate, complete, and expresses the requirements in a manner that can be understood by the system owner. The completion of the draft Software Requirements Specification is an appropriate time to schedule an In-Phase Assessment (IPA).
SDLC Reference:	Appendix D, *In-Phase Assessment Process Guide* provides a description and instructions for conducting an IPA.
Resource:	A template of the Software Requirements Specification document is available on the Research and Policy Web site at: http://www.michigan.gov/dit

Establish Functional Baseline

Responsibility:	Project Manager / Team
Description:	The functional baseline, sometimes called a system requirements baseline, is the main technical deliverable of the Requirements Definition Phase. The system requirements are baselined after the system owner's formal approval of the Software Requirements Specification. Once the requirements are baselined, any changes to the requirements must be managed under change control procedures established in the Software Configuration Management Plan. Approved changes must be incorporated into the Software Requirements Specification.
Deliverables:	Prepare the final Software Requirements Specification and submit to the system owner and customers for their review and approval. The approved Software Requirements Specification is the official agreement and authorization to use the requirements for the software product design. Approval implies that the requirements are understood, complete, accurate, and ready to be used as the basis for the subsequent lifecycle phases. It is important for the system owner/customers to understand that changes to the approved Software Requirements Specification affect the project scope and therefore can change the project cost, resources, or schedule. It is the responsibility of the project manager and project team to identify system owner/customer requested changes that would result in a change of project scope; evaluate the potential impact to the project costs, resources, or schedule; and notify the system owner of the project planning revisions that will be required to accommodate their change requests. Place a copy of the approved Software Requirements Specification in the Project Notebook.
Review Process:	The system owner and customers should review the Software Requirements Specification. After making the changes needed to resolve problems found during the review, the functional baseline is formally established upon receipt of the system owner's approval.

Develop Project Test Plan

Responsibility:

Project Manager

Description:

The Project Test Plan is a narrative and tabular description of the test activities planned for the project during development or enhancement. The Project Test Plan should establish the testing necessary to validate that the project requirements have been met and that the deliverables are at an acceptable level in accordance with existing standards. The plan also ensures that a systematic approach to testing is established and that the testing is adequate to verify the functionality of the software product.

The Project Test Plan includes the resources, project team responsibilities, and management techniques needed to plan, develop, and implement the testing activities that will occur throughout the lifecycle. If individuals outside of the project team perform system and acceptance testing, the plan includes the responsibilities and relationships of external test groups.

In this phase, the plan is written at a high level and focuses on identifying test techniques and test phases. Detailed information about test products (i.e., test plans, test procedures, and test reports) is added to the Project Test Plan as the project progresses through subsequent lifecycle phases.

Development of the Project Test Plan is the responsibility of the project manager. If a test group outside the project team will be involved in any test phase, the project manager must coordinate the Project Test Plan with each test group. The Project Test Plan must be reviewed and approved by the system owner prior to conducting any tests.

Note: For small software projects, a formal Project Test Plan may not be necessary; however, a test approach and testing are required.

Note: Although acceptance testing is part of the project testing, it is discussed separately in more detail.

Deliverables:

When the Project Test Plan is complete, it should contain the following information:

- Describe the occurrence and timing of the test phases in the lifecycle and the entrance and exit criteria for each test phase.
- Specify the test products at each test phase. Describe the types and scope of the testing activities to be performed on each component of the application and the group who is responsible to develop them.
- Map what requirements are verified in what test phase.
- Establish the criteria for evaluating the test results of each test phase.
- Make an initial determination of the resources necessary to accomplish the testing.
- Identify the appropriate person or group to conduct each type of testing activity.

Develop Project Test Plan

Deliverables Continued:

- ❑ Outline the test environment (hardware, software, test tools, and data) needed to conduct the tests.
- ❑ Develop a preliminary schedule for executing the test activities.

Place a copy of the Project Test Plan in the Project Notebook.

Review Process:

Conduct structured walkthroughs to assure the Project Test Plan document adequately describes all testing activities, test schedules, test products, test Responsibilities, the testing lifecycle, and the required resources.

Resources:

Templates for Project Test Plans and Acceptance Test plans are available on the Research and Policy Web site at:

http://www.michigan.gov/dit

Tasks:

Preparation of the Project Test Plan involves the following tasks:

Identify Test Techniques
Identify Test Phases
Identify Test Environment Requirements

Identify Test Techniques

Description:

The Project Test Plan should specify the testing techniques planned for the project including the types of tests required, test documents, test methods, and test data collection. Each test from unit through acceptance testing is specified in terms of entrance and exit criteria and the expected level of involvement from the project team, test group, and other functional areas.

Unit and integration tests with appropriate data must be developed to exercise and validate all specified application requirements, functions, and objectives. System and acceptance tests validate that the integrated system meets the requirements.

Each type of test must use controlled computer generated or live data as specified. The test data must be prepared to include values that will verify the functional capabilities of the software test component, identify its limitations and deficiencies (if any), exercise its capabilities, and verify that the software component performs its intended function as required.

If pilot testing or a phased implementation is required for the software product, the Project Test Plan should include such requirements. In the case of an implementation involving phased software releases, the plan should include the requirements for regression testing of the complete application as new elements are introduced.

For each type of test conducted, the test results are compared with the expected results. Discrepancies are identified and any problems resolved. Retesting is required to verify that the problem solution eliminates the problem and does not introduce new errors. A completed test results/error log form accompanies the final test results. This form is completed by the individual(s) responsible for testing and attached to the documents that certify the completion of each type of test.

Identify Test Phases

Description:

The software product should be tested in four sequential phases: unit, integration, system, and acceptance. Some projects may require additional types of tests (such as prototype testing for offsite installations). The five test phases are described below:

1. Unit Test Phase: The unit test phase involves testing of the individual software units or groups of related units. A unit is a component that is not subdivided into other components; it is a logically separable part of a computer program. Evaluate each unit of code on how well it meets the performance requirements for which it was designed.

Consider timing, memory, accuracy in producing numerical and logical results; and the preparation of input and output required for validating program logic, syntax, and performance requirements. This test phase is performed by the programmer(s) responsible for writing the code.

2. Integration Test Phase: Integration testing is an orderly progression of testing in which software elements, hardware elements, or both are combined and tested to evaluate the interaction between them. Each program/module must be tested. Integration testing is required to validate that groups of related programs, when combined to establish an integrated functional module of code, interface properly, and perform the software functions for which they were designed. Examine the source program/module statements to ensure that the program logic meets the requirements of the design and that the application satisfies an explicit functional requirement. The project team performs this test phase.

3. System Test Phase: The system test phase tests the integrated hardware and software to verify that the software product meets its specified requirements and operates successfully on the host platform. This test phase is required to validate, when the entire software product is loaded onto the host platform, that the proper initialization is performed; decision branching paths are appropriate; and all software functions are performed as specified in the Software Requirements Specification. System testing validates that the software product develops the required outputs and interfaces properly with other systems with which the software product gives or receives data; that transaction response times meet customer expectations; and machine resource allocation and utilization are within expected norms. This test phase can be performed by the project team or by an independent test group with support from the project team.

4. Acceptance Test Phase: Acceptance testing is conducted to determine whether a software product satisfies its acceptance criteria and to enable the system owner's organization to determine whether to accept the software product. The acceptance test is required to validate that the software, its related documentation, tools, and hardware, satisfy all of the specified requirements and objectives of the system owner's organization, SOM and agency standards, the requirements specification, and the design criteria. Acceptance testing will include tests of all intrasystem interfaces; and the use of all manuals, documentation, procedures, and controls. The project team can perform this test phase with system owner and customer observers or by system owner and customer representatives with support from the project team.

Identify Test Phases

Description Continued:

5. Prototype Testing: In addition to the four test phases, a prototype or site test can be used when software must be physically transported, installed, and made operational at a computer facility other than at the site(s) where the acceptance test was conducted. When required, this test is conducted at selected customer location(s) that will totally test the software product under "live" conditions with customers and support personnel.

Note: Unit test, integration test, and system test may be contained within a single Project Test Plan.

Identify Test Environment Requirements

Description:

The Project Test Plan should outline what is needed to perform testing activities throughout the project lifecycle including personnel, hardware, software, space, and other environmental requirements. As much testing as possible should be performed on the same equipment that will be used for the production system. In many cases, this information is not fully known until the System Design Phase.

The following are some of the considerations for test environment requirements:

- Evaluate automated testing tools for the following:
  - Generation of test scripts
  - Creation of result and error repositories
  - Consideration of each tool's benefits and costs
  - Use of simulators
- Determine local area network, wide area network, and metropolitan area network testing environment(s), as needed
- Determine test lab, data generation, and error correction support
- Identify Beta test sites

Develop Acceptance Test Plan

Responsibility:

Project Team

Description:

The Acceptance Test Plan is a description of the test activities planned for project acceptance. The Acceptance Test Plan should establish the testing necessary to validate that the project requirements have been met and that the deliverables are at an acceptable level in accordance with existing standards. The plan also assures that a systematic approach to acceptance testing is established and that the testing is adequate to verify the functionality of the software product.

The complete set of system requirements and acceptance criteria form the basis for determining the overall approach to acceptance testing and the specific testing and examination methods. Features of the installation site and the software system affect how the software acceptance testing will be done. Unique arrangements may be necessary when the software cannot be completely installed and executed in a live environment. Multiple configurations may have to be distributed at several installation sites.

When a new system is a replacement for one already in use, the acceptance test must assure the integrity of the customers business operations while placing the replacement into operation. For example, the old system and the new system are used in parallel until complete functionality has been verified. In some cases, the acceptance process may take several months to assure that a complete business or accounting cycle has occurred. This concern will influence the approach to software acceptance testing.

Deliverables:

Software acceptance testing must be documented carefully with traceability of test cases to the software requirements and acceptance criteria established by the system owner. As a minimum, the acceptance test plan should address the following requirements:

- Identification of the personnel involved in the acceptance test process and their testing responsibilities. If individuals outside of the project team perform acceptance testing, include the responsibilities and relationships of external test groups.
- Traceability of test designs and cases to software requirements.
- The objectives and constraints for each test.
- Complete test cases and test procedures including inputs and expected outputs for each test case.
- Descriptions of error reporting, analysis, and resolution.
- Location(s) where testing will occur, the testing approach, type of facilities, and tester training.
- Acquisition of special purpose testing equipment, tools, and software.
- Resources and cost estimation to accomplish testing.

Develop Acceptance Test Plan

Review Process:

Place a copy of the draft Acceptance Test Plan in the Project Notebook. The draft plan will be reviewed during the Software Integration and Testing Phase and delivered as a final document.

Conduct structured walkthroughs to assure the draft Acceptance Test Plan adequately describes all testing activities, test schedules, test products, test responsibilities, the testing lifecycle, and the required resources.

Resources:

A template of the Acceptance Test Plan is available on the Research and Policy Web site at: http://www..michigan.gov/dit

Select Design Technique

Responsibility:

Project Team

Description:

A systematic approach for building the functional and system designs for the software product simplifies the process and results in a software product that is testable, reliable, and maintainable. A complete design technique includes the following elements:

- A technique that is compatible with the requirements analysis technique and any automated tools used by the project team.
- Simple rules that relate information obtained during requirements analysis to a distinct software structure.
- Design standards that comply with the site's current software development practices, the system owner organization's standards, and the constraints imposed by the software and hardware tools used by the project team.
- A practical approach to design that is agreeable to a wide variety of software products.
- The development of small, intermediate design products that can be used to measure quality and progress.
- An evolution process from functional to system design.
- Well-defined measures to assess the quality of the design.
- Guidance on how to detect and correct design features that reduce maintainability and reusability.

Automated tools that directly support the technique can significantly enhance the value of a design technique. Automated tools provide assistance in generating, maintaining, and analyzing design diagrams and data dictionaries. The use of such tools typically results in a design that is easier to maintain, higher in quality, and more complete than designs developed without automated tools. The increased quality leads to significant productivity gains during software programming and testing.

Deliverables:

Create a description of the design technique and distribute it to the project team, system owner, and customers. Place a copy of the design technique description in the Project Notebook.

Review Process:

Conduct a structured walkthrough to verify that the design technique is appropriate for the scope and objectives of the project. A structured walkthrough is not needed when the technique has been used successfully on similar projects for the same system owner/customer computing environment.

Select Design Technique

Illustration of Design Methods:

The following is an illustration of some common design techniques:

- Function-oriented design methods model the software product by breaking it into components, identifying the inputs required by those components, and identifying the outputs developed by them. Function-oriented design methods include structured analysis and structured design. The major models or design representations used by this method are data flow diagrams, data dictionaries, structure charts, and process specifications.
- Data-oriented design methods use program structures that are derived from the data structures. Tree diagrams are typically used to represent both the data and the program structures.
- Object-oriented design methods develop a software architecture based on the objects manipulated by systems or subsystems rather than by functions. An object-oriented design closely resembles a model of reality since it captures the real-world objects and the operations taken by or upon them. The design structure tends to be layers of abstraction where each layer represents a collection of objects with limited visibility to other layers.

Revise Project Plan

Responsibility:	Project Manager
Description:	Once the requirements are baselined, determine if the project estimates for resources, cost, and schedule need to be revised and if the selected development approach is still appropriate for the size and complexity of the project.
Deliverables:	Review the Project Plan for accuracy and completeness of all Requirements Definition Phase activities and make any changes needed to update the information. Expand the information for the Functional Design Phase to reflect accurate estimates of resources, costs, and hours. ***Note:*** A Project Plan is an effective management tool that is recommended for all projects regardless of size. The plan can be consolidated for small projects.
Review Process:	Conduct a structured walkthrough to assure that the Project Plan reflects the project's current status and adequately estimates the resources, costs, and schedule for the Functional Design Phase. The Project Plan is formally reviewed during the In-Phase Assessment and Phase Exit processes.

Conduct Project Reviews

Conduct Structured Walkthroughs

Responsibility:	Project Manager, Work Product Author and Reviewers
Description:	During the Requirements Definition Phase, schedule at least one structured walkthrough to review each of the Requirements Definition Phase deliverables, i.e., Continuity of Operations Statement/Plan, Software Requirements Specification, Project Test Plan, and draft of the Acceptance Test Plan. ***Note:*** A structured walkthrough is an effective project management tool that is recommended for all projects regardless of size; however, if an item has gone through a prior formal structured walkthrough, and the modifications are minor, a less formal review may be warranted.
SDLC Reference:	*Appendix C, Conducting Structured Walkthroughs*, provides a procedure and sample forms that can be used for structured walkthroughs.

Conduct In-Phase Assessment

Responsibility:	Project Manager and Independent Reviewer
Description:	Schedule at least one IPA prior to the Requirements Definition Phase Exit process. Additional IPAs can be performed during the phase, as appropriate. An IPA is recommended after the completion of the Software Requirements Specification. ***Note:*** An IPA is an effective project management tool that is recommended for all projects regardless of size.
SDLC Reference:	*Appendix D, In-Phase Assessment Process Guide*, provides a procedure and sample report form that can be used for In-Phase Assessments.

Conduct Requirements Definition Phase Exit

Responsibility:	Project Manager
Description:	Schedule the Phase Exit as the last activity of the Requirements Definition Phase. It is the responsibility of the project manager to notify the appropriate participants when a project is ready for the Phase Exit process and to schedule the Phase Exit meeting. All functional areas and the Quality Assurance representative involved with the project should receive copies of the deliverables produced in this phase. ***Note:*** A Phase Exit is an effective project management tool that is recommended for all software projects regardless of size. For small software projects, phases can be combined and addressed during one Phase Exit.
SDLC Reference:	*Appendix E, Phase Exit Process Guide*, provides a procedure and sample report form that can be used for phase exits.

Records Retention and Disposition

Responsibility:

Project Manager, Records Analyst, Developer, Customer Coordinator, Data Administrator

Description:

Record retention and disposition issues must be identified during the requirements definition phase so the system is designed to implement legal record retention requirements.

Deliverables:

Contact the Department of Management and Budget, Records Management Division to have a Retention and Disposal Schedule developed and approved for all related data, inputs, outputs and systems documentation.

- See Procedures 0910.01, 0910.02 and 0920.04.
- See the State of Michigan Records Management Manual. http://www.michigan.gov/hal/0,1607,7-160-17451_18673_19379---,00.html

Identify the legal retention requirements that are imposed upon the system to be developed:

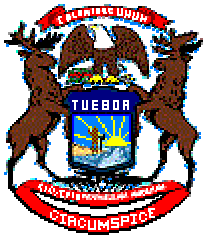
- ❑ What business process does the system support? Do any state or federal laws and/or regulations apply to the data?
- ❑ How is data entered into the system? How long do input documents need be retained after the data is entered into the system?
- ❑ Will legacy data be input into the system? What are the sources of this legacy data?
- ❑ What data and metadata from this system is necessary to provide complete evidence of a transaction? Do any laws or regulations specify the structure (including medium, format, relationships) of the record or transaction or any of its components?
- ❑ What standard outputs does the customer intend to generate? What format will these outputs be in (electronic, paper, COM, COLD, etc.)? How long do the outputs need to be retained?
- ❑ What information (manuals, forms, procedures, etc.) is necessary to interpret the contents of the record?
- ❑ Is any of the data exempt from public disclosure according to FOIA or some other law or regulation?
- ❑ How will the record be reproduced to meet the needs of internal and external customers? What record reproduction formats will be used?
- ❑ How long does data need to reside in the system? If data will be purged, how frequently should this occur?
- ❑ How will data that is eligible for disposition be identified by the system? How will this data be purged from the system so that it cannot be recovered using the active system or the backup system?

Records Retention and Disposition

Deliverables Continued:

- If data will be retained for more than five years (or permanently) it will eventually be necessary to migrate the data to ensure ongoing accessibility of the data? Is migration the best option for ensuring continued accessibility?
- Does the data have permanent value to the creating agency? Has the State Archives of Michigan been consulted to determine if the data has permanent historical value? What documentation will be needed to ensure continued access to the data?
- What portions of the system might be targeted by a lawyer or the Attorney General during litigation or by an auditor?
- Has a formal risk assessment of the system been completed?
- Has a disaster prevention and recovery plan been developed?

DMB 504 "Records Retention and Disposal Schedule"



SYSTEMS DEVELOPMENT LIFECYCLE

SECTION 6

FUNCTIONAL DESIGN PHASE

Table of Contents

Highlights of Phase

Functional Design Phase

Inputs:

- Project Notebook
- Software Configuration Management Plan *(draft)*
- Continuity of Operations Statement/Plan
- Data Dictionary
- Requirements Traceability Matrix
- Software Requirements Specification
- Project Test Plan
- Acceptance Test Plan *(revised)*
- Design methodology
- Project Plan *(revised)*
- Software Quality Assurance Plan

Outputs:

- Design records
- Logical model
- Data Dictionary
- Requirements Traceability Matrix *(expanded)*
- Functional Design Document
- Minutes from Functional Design Review
- Hardware and software procurement records
- Project Plan *(revised)*

Key Activities:

Determine Software Structure
Design Content of System Inputs and Outputs
Design Customer Interface
Design System Interfaces
Design System Security Controls
Build Logical Model
Build Data Model
Develop Functional Design
Initiate Procurement of Hardware and Software
Revise Project Plan
Conduct Structured Walkthrough(s)
Conduct In-Phase Assessment
Conduct Functional Design Phase Exit

Methods and Tools

Define requirements for:

- Data flowing in and out of system - hardcopy, and softcopy
- Processes required for the processing of data - manual and automated
- Storage required - structure of data, relationships between data and size of files

Choose solution considering:

- Technological feasibility
- Cost / Benefit analysis
- Operational feasibility

Forms Utilized

Functional Design Document

Section 6: Functional Design Phase

Overview

Description:

1) *What to do;*
2) *How to do it;*
3) *Define structure of software product;*
4) *Logical system flow;*
5) *Data organization;*
6) *Systems inputs;*
7) *System outputs;*
8) *Processing rules;*
9) *Operational characteristics*

The functional design process maps the "what to do" of the Software Requirements Specification into the "how to do it" of the design specifications. During this phase, the overall structure of the software product is defined from a functional viewpoint. The functional design describes the logical system flow, data organization, system inputs and outputs, processing rules, and operational characteristics of the software product from the customer's point of view. The functional design is not concerned with the software or hardware that will support the operation of the software product, or the physical organization of the data or the programs that will accept the input data, execute the processing rules, and produce the required output.

The focus is on the functions and structure of the components that comprise the software product. The goal of this phase is to define and document the functions of the software product to the extent necessary to obtain the system owner and customers understanding and approval and to the level of detail necessary to build the system design. Prototyping of sys[illegible] functions can be helpful in communicating the design specifications to the system owner and customers. Prototypes can be used to simulate one function, a module, or the entire software product. Prototyping is also useful in the transition from the functional design to the system design.

Review Process:

Quality reviews are necessary during this phase to validate the product and associated deliverables. The activities that are appropriate for quality reviews are identified in this section and the *Section 2 Lifecycle Model.* In addition, a Preliminary Design Review will be conducted. This review is an important milestone in the design process. The time and resources needed to conduct the walkthroughs and Functional Design Review should be reflected in the project resources, schedule, and work breakdown structure.

SDLC References:

Section 2 Lifecycle Model, Quality Reviews, provides an overview of the Quality Reviews to be conducted on a project.

Appendix C, Conducting Structured Walkthroughs, provides a procedure and sample forms that can be used for structured walkthroughs.

Appendix D, In-Phase Assessment Process Guide, provides a procedure and sample report form that can be used for in-phase assessments.

Appendix E, Phase Exit Process Guide, provides a procedure and sample report form that can be used for phase exits.

Determine Software Structure

Responsibility:	Project Team Analysts
Description:	A hierarchical approach is useful for determining the structure and components of the software product. Software system decomposition is one hierarchical approach that divides the software system into different levels of abstraction. Decomposition is an iterative process that continues until single purpose components (i.e., design entities or objects) can be identified. Decomposition is used to understand how the software product will be structured, and the purpose and function of each entity or object. The goal of the decomposition is to create a highly cohesive, loosely coupled, and readily adapted design. A design exhibits a high degree of cohesion if each design entity in the program unit is essential for that unit to achieve its purpose. A loosely coupled design is composed of program units that are independent or almost independent. Several reliable methods exist for performing system decomposition. Select a method that enables the design of simple, independent entities. Functional and object-oriented designs are two common approaches to decomposition. These approaches are not mutually exclusive. Each may be applicable at different times in the design process.
Deliverables:	Refer to each task for applicable deliverables.
Review Process:	Refer to each task for applicable review processes.
Tasks:	The software system decomposition activity includes the following tasks: Identify Design Entities Identify Design Dependencies

Identify Design Entities

Description:

Design entities result from a decomposition of the software product requirements. A design entity is an element (or object) of a design that is structurally and functionally distinct from other elements and is separately named and referenced. The number and type of entities required to partition a design are dependent on a number of factors, such as the complexity of the software product, the design method used, and the programming environment. The objective of design entities is to divide the software product into separate components that can be coded, implemented, changed, and tested with minimal effect on other entities.

Characteristics:

A design entity attribute is a characteristic or property of a design entity. It provides a statement of fact about an entity. The following are common characteristics that should be considered for each design entity.

- Assign a unique name to each entity.
- Classify each entity into a specific type. The type may describe the nature of the entity, such as a subprogram or module; or a class of entities dealing with a particular type of information.
- Describe the purpose or rationale for each entity. Include the specific functional and performance requirements for which the entity was created.
- Describe the function to be performed by each entity. Include the transformation applied to inputs by the entity to produce the desired output.
- Identify all of the external resources that are needed by an entity to perform its function.
- Specify the processing rules each entity will follow to achieve its function. Include the algorithm used by the entity to perform a specific task and contingency actions in case expected processing events do not occur.
- Describe the data elements internal to each entity. Include information such as the method of representation, format, and the initial and acceptable values of internal data. This description may be provided in the data dictionary.

Deliverables:

Maintain a record of all design entities. The records will be integrated into the Functional Design Document. Place a copy of the design entity information in the Project Notebook.

Review Process:

Schedule structured walkthroughs to verify that the design entities are correct, complete, and possess the required characteristics.

Identify Design Dependencies

Description:

Design dependencies describe the relationships or interactions between design entities at the module, process, and data levels. These interactions may involve the initiation, order of execution, data sharing, creation, duplication, use, storage, or destruction of entities.

Identify the dependent entities of the software system design, describe their coupling, and identify the resources required for the entities to perform their function. Also define the strategies for interactions among design entities and provide the information needed to perceive how, why, where, and at what level actions occur.

Dependency descriptions should provide an overall picture of how the software product will work. Data flow diagrams, structure charts, and transaction diagrams are useful for showing the relationship among design entities.

The dependency descriptions may be useful in producing the system integration plan by identifying the entities that are needed by other entities and that must be developed first. Dependency descriptions can also be used to aid in the production of integration test cases.

Deliverables:

Add specific dependency information to the design entity records. The records will be integrated into the Functional Design Document. Place a copy of the dependency information in the Project Notebook.

Review Process:

Schedule structured walkthroughs to verify that the design entities and dependencies are correct, complete, and possess the required characteristics.

Design Content of Systems Input and Output

Responsibility:

Project Team Analysts

Description:

Design the content and format for each of the software product inputs and outputs based on the system input and output requirements identified during the Requirements Definition Phase. Involve the system owner and customers in the design process to make certain that their needs and expectations are being met.

Steps:

Use the following steps to implement the design process:

- ❑ Identify the types of electronic and printed input that will be accepted by the software product, such as data entered manually from source documents and files or records extracted from other systems.
- ❑ Identify the types of electronic and printed output that will be developed by the software product; such as data, records, or files; screen displays; and printed reports. Also identify the output that will be exported to other systems.
- ❑ Identify the specific input and output items that already exist and the items that will be created for input or output as part of the software product.
- ❑ Assign a name to each type of input and output and describe each item from a functional perspective.
- ❑ Identify the system owner/originator of each type of input and output.
- ❑ Identify the frequency of each type of input and output.
- ❑ Design the content and format for each new input and output item or modify the format of existing items that must be changed to accommodate the new software product.

Deliverables:

Document the design for the system inputs and outputs in accordance with the project design standards. Discuss the designs with the system owner and customers and submit completed designs for their review and approval. The approved designs will be incorporated into the Functional Design Document. Place a copy of the system input and output designs in the Project Notebook.

Review Process:

Schedule a structured walkthrough to verify that the system input and output designs are correct and complete.

Design Customer Interface

Responsibility:

Project Team Analysts

Description:

Design a customer interface that is appropriate for the customers, content, and operating environment for the software product. Determine interface levels for all categories of customers. For interactive customer environments, prototype the customer interface. Arrange for customers to experiment with the prototypes so that design weaknesses in the interface can be identified and resolved early. Use prototypes to gain customer acceptance of the interface.

If the site or system owner's organization has an existing customer interface standard, this standard should be used to specify the customer interface for every software product developed for that organization. A customer interface standard should be developed and maintained for each organization that does not have one.

Review the standard each time a new software product is planned to verify that the customer interface is compatible with the software product's selected system architecture. For example, some Windows NT -based customer interface standards would not be appropriate for a Windows 2000-based software product.

Basic Guidelines:

The following basic guidelines can help improve the software product customer interface when there is graphical, command-based, menu-driven, or block mode features:

- Give customers control. Let them choose actions to perform.
- Give customers feedback and progress reports. Tell them when the system is working and when an action is completed.
- Make sure programs, windows, and functions are consistent within and with other components of the software product.
- Be consistent in the format and wording of text.
- Keep it simple. White space is as important on the screen as on the printed page. Reduce screen clutter.
- Use special effects carefully and sparingly. Be sure color screens also work in one color--some customers are colorblind, and some customers have monochrome monitors. Use color consistently. Beeps and other sounds can be annoying; so let customers turn sound off.
- Put information where it can be easily seen; avoid information in corners or borders.
- Limit the amount of information customers must know. Offer choices instead of making customers remember and manually enter information. Provide defaults, and make sure they are logical and satisfy a large number of customers.

Basic Guidelines

- Offer shortcuts. Keyboard shortcuts (e.g., hot keys) and command

Design Customer Interface

Continued:

abbreviations help experienced customers work more quickly.

- Help customers get out of trouble. Provide messages that are understandable and that offer solutions.
- Let customers reverse their actions. If an action will destroy something, identify the object of destruction and wait for a response.

Deliverables:

Refer to each task for applicable deliverables.

Review Procedures:

Refer to each task for applicable review processes.

Tasks

The following tasks are involved in specifying the customer interface:

Design Menu Hierarchy
Design Data Entry Screens
Design Display Screens
Design Online Help
Design System Messages

Design Menu Hierarchies

Description:

Use the following guidelines to improve the design of menu hierarchies:

- Choose an organizing principle for the menu options, such as:
  - Expected frequency of use
  - Logical sequence of operations
  - Alphabetical order (should be used for horizontal word menus with five or more words)
- Put a meaningful title at the top of every menu.
- For full-screen menus, provide symmetric balance by centering the title and the menu options around the center axis of the screen.
- To facilitate scanning, put blank lines between logical groupings of menu options and after about every fifth option in a long list.
- Limit the number of menu choices to one screen.
- Select icons that are intuitive to the function they represent.
- Use a menu option selection method that is consistent with the technology available at the customer's workstation and the size of the software product being designed, such as:
  - Numbers
  - Letters or letter combinations
  - Cursor movement
- Provide a way for the customer to leave the menu without performing any action. Be sure that the option to leave the menu describes the consequences of its selection.
- Words used for menu options should follow these rules:
  - Use words that clearly and specifically describe what the customer is selecting.
  - Use common English words rather than computer or technical jargon. When space permits, spell out words completely.
  - Use simple, active verbs to tell customers what actions will result from their choice. Try to start each option with a verb.
  - Use parallel construction to describe the options.

Design Menu Hierarchies

Description Continued

- Minimize the highlighting used on a menu. Highlighting should be limited to situations where the customer needs to know that there is an exception to the normal practice.
- Do not require the customer to enter leading or trailing blanks or zeros, and do not include a default value on a menu.
- Display the menu options in mixed letters (i.e., upper and lower case).
- Organize menu hierarchies according to the tasks customers will perform, rather than the structure of the software modules.

Deliverables:

Document the design for the menu hierarchy in accordance with the project design standards. Discuss the design with the system owner and customers and submit the completed design for their review and approval. The approved design will be incorporated into the Functional Design Document. Place a copy of the menu hierarchy design in the Project Notebook.

Review Process:

Conduct a structured walkthrough to ensure that the menu hierarchy design is complete and logical.

Design Data Entry Screens

Description:

Use the following guidelines to improve the design of data entry screens:

- When the customer must transcribe data directly from a source document to the screen, the layout of the screen should be similar to the layout of the source document.
- Group data fields into logical categories on the screen; provide a header that describes the contents of each category.
- Make areas of the screen that are not needed for data entry or commands inaccessible to the customer.
- Do not require the customer to enter information that is already available to the software or can be computed by it.
- Do not require the customer to enter dimensional units, leading or trailing blanks, or zeros.
- Allow the customer to enter data by character replacement.
- Put a caption describing the data to be entered adjacent to each data field; incorporate memory joggers into the caption.
- Justify data entries automatically.
- Display default values in data fields when appropriate.
- Provide context-sensitive help for data entry fields.

Deliverables:

Document the designs for the data entry screens in accordance with the project design standards. Discuss the design with the system owner and customers and submit the completed designs for their review and approval. The approved designs will be incorporated into the Functional Design Document. Place a copy of the data entry screen designs in the Project Notebook.

Review Process:

Conduct a structured walkthrough to assure that the data entry screen designs are consistent, complete, and logical.

Design Display Screens

Description:

Use the following guidelines to design display screens that are easy to use and understand:

- Put a title on every display screen. The title should clearly and specifically describe the contents of the screen.
- Display only information that the customer needs to know.
- Display data to the customer in directly usable form.
- Provide symmetric balance to displays by centering titles and headings and by placing information on both sides of the center axis.
- Every display should indicate how to exit from the screen. Use consistent exit procedures.
- When the display continues over multiple screens, the screen should indicate where the customer is in the display (e.g., Screen 1 of 3).
- Data fields need to be grouped into logical categories or according to the structure of a source document (when there is one).
- Be consistent in the use of words and special characters.
- Display text conventionally in mixed letters (i.e., upper and lower case) and with appropriate punctuation. Avoid all uppercase letters. Put a blank line between paragraphs.
- Left justify text, and leave a ragged right margin.
- Avoid hyphenation of words between lines.
- Use abbreviations and acronyms only when they are significantly shorter than the full text and when the customer will understand them.
- Be consistent with the format of information being displayed.
- Consider the skills of the customers and the information they will manipulate when information is displayed in multiple windows.

Table and List Guidelines:

Use the following guidelines to improve the design of online tables and lists:

- Put a meaningful label on the columns and, if appropriate, the rows of tables and lists. Continue the labels when a table or list extends over more than one screen.
- If data items are continued on subsequent screens, the labels should be added to each screen.

Design Display Screens

Table and List Guidelines Continued:

- If data items are scrolled, the labels should be fixed on the screen and not be part of the scrolled area (they remain in place as the body of the table or list changes).
- Arrange the items in a table or list in some recognizable order to facilitate scanning.
- Put items in a multiple column list in vertical columns that are read from left to right on the screen.
- Left justify columns of alphabetic data; right justify columns of numeric data or align them by the decimal point or other delimiter.
- Insert a blank line after about every fifth row in a long column.
- Insert a minimum of two spaces between the longest item in a column and the beginning of the next column.
- When listed items are labeled by number start with a one (1) not a zero (0).

Deliverables:

Document the design for the display screens in accordance with the project design standards. Discuss the designs with the system owner and customers and submit the completed designs for their review and approval. The approved designs will be incorporated into the Functional Design Document. Place a copy of the display screen designs in the Project Notebook.

Review Process:

Conduct a structured walkthrough to ensure that the display screen designs are consistent, complete, and logical.

Design Online Help

Description:

Online help is typically requested by customers when they want to perform a new, complex, or infrequently used procedure, or when they do not know what else to do. The text of online help messages needs to be planned, drafted, and evaluated as carefully as print documentation. In addition, the layout and format of online help must be designed to deal with the special constraints imposed by the video screen.

Use online help to explain concepts, procedures, messages, menu choices, commands, words, function keys, and formats. Work with the customers to identify the level of detail needed for online help. Determine whether the customers need a one-line message at the bottom of the screen or a full online explanation with successive levels of detail.

Effective online help messages tell customers what the software product is doing, where they are in the sequence of screens, what options they have selected, and what options are available.

Guidelines:

The following guidelines can improve the design of online help:

- ❑ Write online help messages in plain English.
  - Straightforward and reads as if it were spoken.
  - Clear, direct, and simple.
  - Effectively organized with a concern for what customers need to know.
- ❑ Address the customer directly as "you"; use the active voice.
- ❑ Use simple action verbs to describe procedures. Do not use nouns to replace pronouns, verbs, and adjectives.
- ❑ Describe procedures in logical order.
- ❑ Provide a direct route back to the function or task being performed.
- ❑ Whenever possible display help text on the screen with the function or task that is being performed.

Deliverables:

Document the design for online help in accordance with the project design standards. Discuss the design with the system owner and customers and submit the completed design for their review and approval. The approved design will be incorporated into the Functional Design Document. Place a copy of the online help design in the Project Notebook.

Review Process:

Conduct a structured walkthrough to ensure that the online help design is consistent, complete, and logical.

Design System Messages

Description:

System messages are the various types of information that the system provides to the customer such as status messages, prompts, and error messages.

Status Messages: Status messages are important for giving customers the feeling they are in control of the software. They tell customers what the software is doing, where they are in the sequence of screens, what options they have selected, and what options are available.

Prompts: Prompts inform the customer to type data or commands or to make a simple choice:

- Use prompts to ask the customer to make a simple choice or to enter data or commands. Be as specific as possible.
- Include memory aids in the prompt to help customers type a response in the proper format and order, initiate infrequently used processes, or clearly identify exceptions to normal practice.
- When defaults are allowed with prompts, indicate clearly which default value will be initiated.

Error Messages: Error messages should allow customers to recover from mistakes by making it clear what the mistake was and how to correct it. Error messages need to be specific about why a mistake was made:

- Design the software product to check for obvious errors.
- Be as specific as possible in describing the cause of an error. Do not use error codes.
- Do not assign blame to the customer or the software in an error message. Use a neutral tone.
- Whenever possible, the error message should indicate what corrective action the customer needs to take.
- Be consistent in the format, wording, and placement of messages.
- Consider describing error messages at more than one level of detail.

Deliverables:

Document the design for the system messages in accordance with the project design standards. Discuss the designs with the system owner and customers and submit the completed designs for their review and approval. The approved designs will be incorporated into the Functional Design Document. Place a copy of the system message designs in the Project Notebook.

Review Process:

Conduct a structured walkthrough to ensure that the system message designs are consistent, complete, and logical.

Design System Interface

Responsibility:	Project Team Analysts
Description:	Develop a design depicting how the software product will interface with other systems based on the system interface requirements identified in the Requirements Definition Phase. Submit the applicable interface designs for review by the system owner or system administrator for each system that will interface with the software product. Any incompatibilities with the interfaces will be identified early in the design process and corrective actions can be initiated to assure each interface is properly designed and coded.
Sample Issues:	The following list provides some of the issues that should be considered when designing the system interfaces: ❑ System inputs and outputs ❑ Method of interface ❑ Volume and frequency of data ❑ Platform of interfacing system ❑ Format of data ❑ Automatic or manual initiation of interface ❑ Need for polling device(s) ❑ Verification of data exchange ❑ Validation of data
Deliverables:	Document the design(s) for the system interfaces in accordance with the project design standards. Discuss the designs with the system owner and customers and submit completed designs for their review and approval. The approved designs will be incorporated into the Functional Design Document. Place a copy of the system interface designs in the Project Notebook.
Review Process:	Schedule a structured walkthrough to verify that the system interface designs are correct and complete.

Design System Security Controls

Responsibility:	Project Team Analysts and Security Personnel
Description:	Design the security controls that will be incorporated into the software product based on the security and access requirements identified during the Requirements Definition Phase. Design the security controls in conjunction with the site or system owner organization's security officer.
Steps:	Use the following step to implement the design process: ❑ Identify the customers and organizations that will have access to the software product. Indicate what access restrictions they will have. All persons in a work area may not have the same security access level. Measures should be taken to assure that unauthorized individuals do not access sensitive materials and software requiring protection. ❑ Identify controls for the software product, such as the customer identification code for system access and the network access code for the network on which the software product will reside. ❑ Identify whether access restrictions will be applied at the system, subsystem, transaction, record, or data element levels. Classified information must be protected in accordance with agency directives. ❑ Identify physical safeguards required to protect hardware, software, or information from natural hazards and malicious acts. ❑ Identify communications security requirements.
Deliverables:	Document the design for the system security controls in accordance with the project design standards. Discuss the design with the system owner and customers and submit the completed design for their review and approval. The approved design will be incorporated into the Functional Design Document. Place a copy of the system security control design in the Project Notebook.
Review Process:	Schedule a structured walkthrough to verify that the system security controls are correct and complete. Include the security officer in the walkthrough.

Build Logical Model

Responsibility:

Project Team Analysts

Description:

The logical model defines the flow of data through the software system and determines a logically consistent structure for the software. Each module that defines a function is identified, interfaces between modules are established, and design constraints and limitations are described. The focus of the logical model is on the real-world problem or need to be solved by the software product.

A logical model has the following characteristics:

- Describes the final sources and destinations of data and control flows crossing the system boundary rather than intermediate handlers of the flows.
- Describes the net transfer of data across the system boundary rather than the details of the data transfer.
- Provides for data stores only when required by an externally imposed time delay.

When building a logical model, the organization of the model should follow the natural organization of the software product's subject matter. The names given to the components of the model should be specific. The connections among the components of the model should be as simple as possible.

Deliverables:

The logical model should be documented in customer terminology and contain sufficient detail to obtain the system owner's and customers' understanding and approval. Use data flow diagrams to show the levels of detail necessary to reach a clear, complete picture of the software product processes, data flow, and data stores.

Maintain the logical model and data flow diagrams for incorporation into the Functional Design Document. Place a copy of the logical model and data flow diagrams in the Project Notebook. Keep the logical model and diagrams up-to-date. They will serve as a resource for planning enhancements during the Maintenance Phase, particularly for enhancements involving new functions.

Review Process:

Schedule a structured walkthrough to verify that the logical model is correct, logical, and complete.

Build Data Model

Responsibility:

Project Team Analysts

Description:

A data model is a representation of a collection of data objects and the relationships among these objects. The data model is used to provide the following functions:

- Transform the business entities into data entities.
- Transform the business rules into data relationships.
- Resolve the many-to many relationships as intersecting data entities.
- Determine a unique identifier (keys) for each data entity.
- Add the characteristics (facts) for each data entity.
- Document the integrity rules required in the model.
- Determine the data accesses (navigation) of the model.

Deliverables:

The data dictionary started in the Requirements Definition Phase is expanded in this phase to catalog every known data element used in the customer's work and every system-generated data element. Data elements are documented in detail to include characteristics, known constraints, input sources, output destinations, and known formats.

The data dictionary can serve as a central repository of information for both programmers and end customers. The dictionary can include business rules, processing statistics, and cross-referencing information for multiple vendor environments.

To expand the data dictionary, define, analyze, and complete data definitions using the following steps:

- Identify data needs associated with various system features.
- Match (verify) data needs with the data dictionary.
- Match the data dictionary with specific data structures.

- Create data record layouts.
- Ensure that all data can be maintained through add, change, or delete functions.

The data dictionary is further refined in the System Design Phase to complete the information on data elements, entities, files, physical characteristics, and data conversion requirements.

Build Data Model

Sample Characteristics:

The following is a sample of the type of characteristics (information) that should be included for each element in a data dictionary:

- ❑ Long data name (full name)
- ❑ Short data name (abbreviation)
- ❑ Alias
- ❑ Data definition
- ❑ Owner(s)
- ❑ Occurrence(s)/key
- ❑ Program mode
- ❑ Input source(s); e.g., screens, external interfaces, system generated
- ❑ Output destination(s); e.g., screens, reports, external interfaces
- ❑ Values/meanings
- ❑ Protection/security
- ❑ Default value
- ❑ Length/precision
- ❑ Character set (type)
- ❑ Format
- ❑ Range
- ❑ Surface edits
- ❑ Remarks

Review Process:

Schedule a structured walkthrough to verify that the data dictionary is correct and complete. The data model for a software application should be validated against any Agency or site-specific data model.

Develop Functional Design

Responsibility:

Project Team

Description:

The software functional design describes how the software product will be structured to satisfy the requirements identified in the Software Requirements Specification. It is a description of the software structure, components, interfaces, and data necessary before coding can begin.

The software functional design is a model or representation of the software product that is used primarily for communicating software design information to facilitate analysis, planning, and coding decisions. It represents a partitioning of the software system into design entities and describes the important properties and relationships among those entities. Design descriptions may be developed as documents, graphic representations, formal design languages, records in a data base management system, and CASE tool dictionaries.

Within the functional design, the design entities can be organized and presented in any number of ways. The goal of this activity is to compile the design entities and their associated characteristics in a manner that facilitates the access of design information from various viewpoints (e.g., project management, configuration management, quality assurance, and testing). Also, the design entities and their characteristics must be described in terms that are understandable to the system owner and customers.

Deliverables:

Each requirement identified in the Software Requirements Specification must be traceable to one or more design entities. This traceability ensures that the software product will satisfy all of the requirements and will not include inappropriate or extraneous functionality. Expand the Requirements Traceability Matrix developed in the Requirements Definition Phase to relate the functional design to the requirements. Place a copy of the expanded matrix in the Project Notebook. Refer to each task for other applicable deliverables.

Review Process:

Conduct a structured walkthrough of the Requirements Traceability Matrix.

Refer to *Section 6, Conduct Functional Design Review*, for the review process.

Tasks:

The following tasks are involved in developing the functional design:

Develop Functional Design Document
Conduct Functional Design Review

Develop Functional Design Document

Description:	The Functional Design Document defines the functions of the system in customer terminology and provides a firm foundation for the development of the system design. The Functional Design Document should be written from the system owner/customers' perspective. This document provides the owner/customers with an opportunity to review and provide input to the software product design before system design work is completed.
Deliverables:	Prepare a draft Functional Design Document. Use the designs developed for inputs, outputs, customer and system interfaces, and security controls as input to this document. Submit the draft document to the system owner and customers for their review and approval. After making the changes needed to resolve problems found during the review, the approved Functional Design Document becomes an official agreement and authorization to use the functional design as the basis for developing the system design. Place a copy of the approved Functional Design Document in the Project Notebook.
Review Process:	Conduct structured walkthroughs as needed to assure that the Functional Design Document is accurate, complete, and describes the functional design in a manner that can be understood by the system owner and customers. The completion of the draft Functional Design Document is an appropriate time to schedule an In-Phase Assessment (IPA).
SDLC Reference:	*Appendix D, In-Phase Assessment Process Guide* provides a description and instructions for conducting an IPA.
Resource:	A template of the Functional Design Document is available on the Web site at: http://www.michigan.gov/dit.

Conduct Functional Design Review

Description:

The Functional Design Review is a formal technical review of the basic design approach. The primary goal of the Functional Design Review is to demonstrate the ability of the software design to satisfy the project requirements. The review should be a series of presentations by the project team to the system owner, customers, functional area points-of-contact, and Quality Assurance representative.

Conduct the Functional Design Review to perform the following verifications:

- Evaluate the progress, technical adequacy, and risk resolution of the selected design approach. Determine whether the project team is following the approved design approach.

- Evaluate the progress, technical adequacy, and risk resolution of the selected test approach. Review the following items:
  - Organization and responsibilities of group conducting tests
  - Project Test Plan
  - Planned format, content, and distribution of test reports
  - Planned resolution of problems and errors identified during testing
  - Retest procedures
  - Change control and configuration management of test items
  - Special test tools not required as deliverables

- Evaluate the techniques to be used to meet quality assurance requirements.

- Establish the existence and compatibility of the physical and functional interfaces.

- Determine whether the functional design embodies all of the software product requirements.

- Verify that the design represents software that can meet the functional, data, and interface requirements.

- Review the planned customer interfaces to the software. Examples of the types of design information to review:

  - Operating modes for each display station. For each mode, the functions performed, the displays and controls used.
  - The format and content standards for each screen (e.g., data locations, spaces, abbreviations, the number of digits, all special symbols, alert mechanisms).
  - Control and data entry devices and formats (e.g., keyboards, special function keys, and cursor control).
  - The format of all data inputs and provisions for error detection and correction.
  - The format for all status and error messages and data printouts (e.g., formats, headings, data units, abbreviations, spacing, columns).

Conduct Functional Design Review

Description Continued:

- Demonstrate any rapid design prototypes used to make design decisions.
- Identify potential high-risk areas in the design and any requirements changes that could reduce risk.
- Review to assure that consideration has been given to optimizing the maintainability and maintenance aspects of the software product.

Review Items:

The following items should be considered for review and evaluation during the Functional Design Review. Be prepared to discuss in technical detail any of these items within the scope of the review:

- **Functional flows**. Indicate how the computer software functional flows map the software and interface requirements to the individual high-level components of the software product.
- **Storage allocation data**. Describe the manner in which available storage is allocated to individual software components. Timing, sequencing requirements, and relevant equipment constraints used in determining the allocation should be included.
- **Control functions.** Describe the executive control and start/recovery features of the software product.
- **Component structure.** Describe the high-level structure of the software product, the reasons for choosing the components, the development technique that will be used within the constraints of available computer resources, and any support programs that will be required in order to develop and maintain the software product and allocated data storage.
- **Security.** Identify the security requirements and provide a description of the techniques to be used for implementing and maintaining security within the software product.
- **Computer facilities.** Describe the availability, adequacy, and planned utilization of the computer software facilities.
- **Computer facility versus the operational system.** Describe any unique design features that exist in the functional design in order to allow use within the computer software engineering facility that will not exist in the operational software product. Provide information on the design of support programs not explicitly required for the operational system that will be generated to assist in the development of the software product.
- **Development tools.** Describe any special tools (e.g., simulation, data reduction, or utility tools) that are not deliverables, but are planned for use during software development.
- **Test tools.** Describe any special test systems, test data, data reduction tools, test computer software, or calibration and diagnostic software that are not deliverables, but are planned for use during software development.

Conduct Functional Design Review

Review Items Continued:

- **Commercial resources.** Describe commercially available computer resources, including any optional capabilities (e.g., special features, interface units, special instructions, controls, formats). Identify any limitations of commercially available equipment (e.g., failure to meet customer interface, safety, and maintainability requirements) and identify any deficiencies.
- **Existing documentation.** Maintain a file and have available for review any existing documentation supporting the use of commercially available computer resources.
- **Support resources.** Describe the resources necessary to support the software product during engineering, installation, and operational state (e.g., operational and support hardware and software personnel, special skills, human factors, configuration management, testing support, documentation, and facilities/space management).
- **Standards.** Describe any standards or guidelines that must be followed.
- **Operation and support documentation.** Describe the documentation that will be developed to support the operation and maintenance of the software product.

Deliverables:

Create and distribute official meeting minutes for each session. The minutes should consist of significant questions and answers, action items and individual/group responsible, deviations, conclusions, and recommended courses of action resulting from presentations or discussions. Recommendations that are not accepted should be recorded along with the reason for non-acceptance. Minutes must be distributed to the system owner and customers for review and notification of review performance as follows:

- **Approval** - indicates that the functional design is satisfactorily completed.
- **Contingent Approval** - indicates that the functional design is not considered accomplished until the satisfactory completion of resultant action items.
- **Disapproval** - indicates that the functional design is inadequate. Another Functional Design Review is required.

Review Process:

Not applicable.

Initiate Procurement of Hardware/Software

Responsibility:

Project Manager/Team

Description:

Careful consideration should be given to purchasing off-the-shelf software before expending the time, resources, and costs associated with developing custom-built systems. Whenever possible, acquire off-the-shelf software to satisfy some or all of the project requirements. In addition, some projects may require the acquisition of hardware or software to support the design, code, and test processes, (see *Appendix I- COTS*.)

Try to acquire a demonstration package of any proprietary software before completing the design specifications. The proprietary software may prove inadequate or inappropriate once it has been evaluated through hands-on use. Create a pilot of the software product to exercise the most important functions provided by the proprietary software as well as to obtain definite performance indications.

Initiate the procurement of any hardware or software well in advance of the planned need for these products. Adequate time must be allocated in the Project Plan timeline for the selection, procurement, installation, testing, and training associated with each vendor product.

The project team may assume all of the procurement, installation, and testing responsibilities, or the functional area that is most familiar with the product may initiate the acquisition and testing of some hardware and software. For example, a local area network engineering group may procure and test local area network or client/server software; a mainframe systems group may procure and test mainframe software.

Note: When the expected operating platform for a software product will require extensive procurement of hardware and software, it is recommended that procurement needs be addressed as early in the lifecycle as possible. If hardware and software acquisition requirements are known, develop the Acquisition and Installation Plans for all operating sites and initiate the procurement process. Review and, if necessary, revise the Production Platform Acquisition and Installation Plans at the beginning of the Programming Phase. Requirements for the Production Platform Acquisition and Installation Plans are provided in Section 8*, Programming Phase.*

Deliverables:

Place a copy of all software and hardware procurement records (e.g., justifications, approvals, purchase orders, and invoices) and the Acquisition and Installation Plans (if developed) in the Project Notebook.

Review Process:

Not required; however, a peer review of software and hardware procurement records can be beneficial to ensure the correct order is placed.

Revise Project Plan

Responsibility:	Project Manager
Description:	Once the Functional Design Review is completed and the functional design is baselined, determine if the project estimates for resources, cost, and schedule need to be revised and if the selected design approach is still appropriate for the size and complexity of the project.
Deliverables:	Review the Project Plan for accuracy and completeness of all Functional Design Phase activities and make any changes needed to update the information. Expand the information for the System Design Phase to reflect accurate estimates of resources, costs, and hours. ***Note:*** A Project Plan is an effective management tool that is recommended for all projects regardless of size. The plan can be consolidated for small projects.
Review Process:	Conduct a structured walkthrough to assure that the Project Plan reflects the project's current status and adequately estimates the resources, costs, and schedule for the System Design Phase. The Project Plan is formally reviewed during the In-Phase Assessment and Phase Exit processes.

Conduct Functional Design Phase Exit

Conduct Structured Walkthroughs

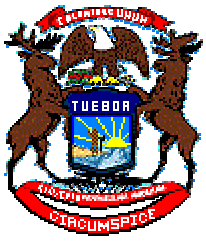
Responsibility:	Project Manager, Work Product Author and Reviewers
Description:	During the Functional Design Phase, schedule at least one structured walkthrough to review each of the Functional Design Phase deliverables, i.e., Logical Model, Data Dictionary, Requirements Traceability Matrix, and Functional Design Document. ***Note:*** A structured walkthrough is an effective project management tool that is recommended for all projects regardless of size; however, if an item has gone through a prior formal structured walkthrough, and the modifications are minor, a less formal review may be warranted.
SDLC Reference:	*Appendix C, Conducting Structured Walkthroughs*, provides a procedure and sample report form that can be used for phase exits.

Conduct In-Phase Assessment

Responsibility:	Project Manager and Independent Reviewer
Description:	Schedule at least one IPA prior to the Functional Design Phase Exit process. Additional IPAs can be performed during the phase, as appropriate. The completion of the Functional Design Document is an appropriate time to schedule an IPA. ***Note:*** An IPA is an effective project management tool that is recommended for all projects regardless of size.
SDLC Reference:	*Appendix D, In-Phase Assessment Process Guide*, provides a procedure and sample report form that can be used for In-Phase Assessments.

Conduct Functional Design Phase Exit

Responsibility:	Project Manager
Description:	Schedule the Phase Exit as the last activity of the Functional Design Phase. It is the responsibility of the project manager to notify the appropriate participants when a project is ready for the Phase Exit process and to schedule the Phase Exit meeting. All functional areas and the Quality Assurance representative involved with the project should receive copies of the deliverables produced in this phase. ***Note:*** A Phase Exit is an effective project management tool that is recommended for all software projects regardless of size. For small software projects, phases can be combined and addressed during one Phase Exit.
SDLC Reference:	*Appendix E, Phase Exit Process Guide*, provides a procedure and sample report form that can be used for phase exits.



SYSTEMS DEVELOPMENT LIFECYCLE

SECTION 7

SYSTEM DESIGN PHASE

Section 7: System Design Phase

Table of Contents

Highlights of Phase

Systems Design Phase Highlights

Inputs:

- Project Notebook
- Design records
- Logical model
- Data dictionary *(expanded)*
- Requirements Traceability Matrix *(expanded)*
- Functional Design Document
- Hardware and software procurement records
- Project Plan *(revised)*
- Software Quality Assurance Plan

Outputs:

- Design specifications
- Physical Model
- Data Dictionary *(expanded)*
- Integration Test Plan *(draft)*
- System Test Plan *(draft)*
- Conversion Plan
- Requirements Traceability Matrix *(expanded)*
- System Design Document
- Program Specifications
- Programming Standards
- Project Plan *(revised)*

Key Activities:

Select System Architecture
Design Specifications for Software Modules
Design Physical Model and Data Base Structure
Develop Integration Test Plan
Develop System Test Plan
Develop Conversion Plan
Develop System Design
Develop Program Specifications
Define Programming Standards
Revise Project Plan
Conduct Structured Walkthrough(s)
Conduct In-Phase Assessment
Conduct System Design Phase Exit

Forms Utilized

Conversion Plan
System Design Document

Methods and Tools

Define procedures and man-machine boundaries:

- Manual and automated processing
- Security and quality issues
- Input and output functions

Tools:

- project management tools
- programming code and hardware
- prototyping tools
- CASE (computer aided software engineering)

Evaluate design

- *Walkthrough* procedures being developed
- Use feedback from users where possible

Present design documentation

- Executive summary of system
- Description and explanation of design decisions
- Detailed documentation required for programmers, storage requirements etc
- Control descriptions

Overview

Description:

1) Translate customer-oriented functional design specifications;
2) *Design the data structure and processes;*
3) *Develop general module specifications;*
4) *Develop system design*

The goal of this phase is to translate the customer-oriented functional design specifications into a set of technical, computer-oriented system design specifications; and to design the data structure and processes to the level of detail necessary to plan and execute the Programming and Installation Phases. General module specifications should be developed to define what each module is to do, but not how the module is to be coded. Effort focuses on specifying individual routines and data structures while holding constant the software structure and interfaces developed in the previous phase. Each module and data structure is considered individually during detailed design with emphasis placed on the description of internal and procedural details. The primary deliverable of this phase is a software system design that provides a blueprint for the coding of individual modules and programs.

Review Process:

Quality reviews are necessary during this phase to validate the product and associated deliverables. The activities that are appropriate for quality reviews are identified in this section and the *Section 2 Lifecycle Model.* In addition, a Critical Design Review is conducted once the System Design Document is developed. This review is an important milestone in the design process. The time and resources needed to conduct the walkthroughs and Critical Design Review should be indicated in the project resources, schedule, and work breakdown structure. *Section 2 Lifecycle Model, Quality Reviews*, provides an overview of the Quality Reviews to be conducted on a project.

SDLC References:

Appendix C, Conducting Structured Walkthroughs, provides a procedure and sample forms that can be used for structured walkthroughs.

Appendix D, In-Phase Assessment Process Guide, provides a procedure and sample report form that can be used for in-phase assessments.

Appendix E, Phase Exit Process Guide, provides a procedure and sample report form that can be used for phase exits.

Select System Architecture

Responsibility:

Project Team

Description:

When the system architecture for the software product has not been predetermined by the existing computing environment of the system owner and customers, evaluate system architecture alternatives to determine which one has the best, cost-effective solution that satisfies the project requirements.

1) *Evaluate system architecture alternatives;*
2) *Select specific hardware, software, database management system and communications facilities;*
3) *Obtain support from functional area points-of-contact*

"Cost effective solution" does not imply the least expensive alternative. The "best, cost effective solution" is the alternative that does the best job of satisfying the project requirements, assures the highest quality software product, and provides for an adequate return on investment in a timeframe that is acceptable to the system owner:

Select the specific hardware, software, data base management system, and communication facilities based on the following types of considerations:

- Agency or site-specific information architecture guidelines or standards
- Hardware and software that emphasizes simplicity, flexibility, ease of operation and maintenance
- Cost to procure and maintain potential environment
- Backup and recovery procedures
- Selection of a distributed or centralized processing environment
- Communication requirements
- Data configuration

Obtain support from functional area points-of-contact to aid in the architecture evaluation process. Consultations and input may be helpful from system and data base administrators, local area network administrators, operations personnel, system programmers, and communication experts.

Deliverables:

Refer to each task for applicable deliverables.

Review Process:

Refer to each task for applicable review processes.

Tasks:

The following tasks are involved in selecting a system architecture:

Evaluate System Architecture Alternatives
Recommend System Architecture

Evaluate System Architecture Alternatives

Description:

Consider system architecture alternatives within the site's information architecture guidelines that enable the project objectives and requirements to be achieved. The selection of system architecture depends on many factors such as the experience of the project team with each alternative and the availability of reusable components to facilitate the implementation of an alternative.

When investigating alternatives, consider the following issues:

- Those functions or portions of functions that are to be automated and the functions that will be manual. Conduct an examination of *what* the automated portion of the project will encompass.
- The technical solution for the objectives. The determinations of *how* the software product is to be designed; (e.g., online vs. batch, client-server vs. mainframe, Oracle vs. SQL Server).
- The system owner and customers' computing environment and the needs created by the technical solution. Consider any hardware and software that must be acquired, including system access software, operating system software, data base management system, and communications facilities.

The following procedure provides one approach for evaluating the architecture Alternatives:

- Conduct an Analysis of Benefits and Costs to determine the most cost-effective alternative. On the benefits side, include the improvements over the current process being used to support the business application. On the cost side, include any degradation from current capabilities along with the rationale for allowing the degradation.
- Create and evaluate a data flow diagram for each alternative.
- Identify how customers would interact with the features associated with each alternative (such as the generation of queries and reports).
- Create a list of the risks associated with each alternative and develop a plan for mitigating each risk.
- Compare the performance capabilities of each alternative. How fast will each alternative be able to process the customer's work given a particular hardware resource. Performance is usually expressed in terms of throughput, run time, or response time. Five factors that frequently affect performance include:
  - Number of intermediate files in a system (park data between programs)
  - Number of times a given file is passed
  - Number of seeks against a disk file
  - Time spent in calling programs and other system overhead
  - Time taken to execute actual program

Evaluate System Architecture Alternatives

Description Continued:

- Compare the security and access control features of each alternative. To what extent does the alternative provide security against human errors, machine malfunction, or deliberate mischief. Some common controls include:

  - Check digits on predetermined numbers
  - Batch control totals
  - Creation of journals and audit trails
  - Limited access to files

- Compare the ease with which each alternative allows the system to be modified to meet changing requirements, such as:
  - Fixing errors
  - Changing customer needs
  - Mandatory/statutory modifications
  - Enhancements

Deliverables:

Maintain records on each alternative that is evaluated. Use this information to develop a summary of the system architecture alternatives. The summary will be integrated into the materials presented to the system owner when a system architecture recommendation is made. Place a copy of the records for each alternative and the summary in the Project Notebook.

If a Cost Benefit Analysis (CBA) is conducted, prepare a report that describes the process used for the analysis, a summary of the alternatives considered, and the results obtained and place a copy in the Project Notebook. The report will be integrated into the materials presented to the system owner when a system architecture recommendation is made.

Review Process:

Conduct structured walkthroughs on records of each alternative that is evaluated.

PMM Reference:

For more information on conducting the Cost Benefit Analysis, refer to, *Conduct Cost Benefit Analysis in Section 3, Project Phase of the Project Management Methodology on Web site at:* http://www.michigan.gov/dit.

Recommend System Architecture

Description:

1) Develop recommendation for system architecture

Based on the results of the architecture alternatives evaluation, develop a recommendation for a system architecture that is cost-effective and will facilitate the achievement of the software project requirements. Prepare a presentation for the system owner and customers that provides the following types of information to support the recommendation:

- ❑ Review the limitations or problems with any current manual or automated system that will be resolved by the software product.
- ❑ Present the logical model for the software product. Highlight new functions that would be incorporated.
- ❑ For each architecture alternative that was evaluated, present the following information:
  - A description of the alternative.
  - An overall data flow diagram showing how the alternative would be implemented.
  - The way the system would look to the customers, in terms of hardware, customer interface, reports, and query facilities.
  - The estimated benefits of the alternative.
  - The estimated cost and time to implement the alternative.
  - A statement of the element of risk associated with the alternative.
- ❑ Present the recommended alternative and explain why it was selected.

Before the project proceeds, the system owner should make a decision about the system architecture either by formally accepting the project team's recommendation or by directing the team to use a different architecture. Any delay in making this decision could result in a slippage of the project schedule.

Deliverables:

Document the project team's recommendations for the most cost-effective and viable architecture alternative. Provide a summary of each alternative that was evaluated. Describe the rationale for proposing the recommended architecture. Describe the impact of this alternative on the system owner and customer's organization(s) and other systems. Include any background information that was relevant to the decision process, such as the Cost Benefit Analysis Report.

Present the project team's recommendation for the system architecture to the system owner and customers. The recommendation can be delivered as a document or as a presentation. Place a copy of the document or presentation materials in the Project Notebook.

Review Process:

Conduct a structured walkthrough to assure that the most cost-effective and viable architecture alternative is being recommended.

Design Specifications for Software

Responsibility:

Project Team

Description:

During the Functional Design Phase, a decomposition of the software product requirements resulted in a collection of design entities (or objects). In the System Design Phase, these design entities are grouped into the routines, modules, and programs that need to be developed or acquired as off-the-shelf or reusable software.

1) Collect design entities;
2) Group design entities;
3) Detail the design

Expand the functional design to account for each major software action that must be performed and each data object to be managed. Detail the design to a level such that each program represents a function that a programmer will be able to code.

Steps:

Use the following steps to design the software module specifications:

- ❑ Identify a software program for each action needed to meet each function or data requirement in the Software Requirements Specification and the data dictionary.
- ❑ Identify any routines and programs that may be available as reusable code or objects from existing applications or off-the-shelf software. Identify programs that must be designed and developed (custom-built). Assign a name to each program and object that is functionally meaningful.
- ❑ Identify the system features that will be supported by each program.
- ❑ Specify each program interface. Update the data dictionary to reflect all program and object interfaces changed while evolving from the functional to the system design.
- ❑ Define and design significant characteristics of the programs to be custom-built.
- ❑ Expand the program interfaces to include control items needed for design validity (e.g., error and status indicators).
- ❑ Combine similar programs and objects. Group the design entities into modules based on closely knit functional relationships. Formulate identification labels for these modules.
- ❑ Show dependencies between programs and physical data structures (e.g., files and global tables). Avoid defining a program that not only needs data residing in a file or global table, but also depends on the physical structure or location of data.
- ❑ Change the design to eliminate features that reduce maintainability and reusability (i.e., minimize coupling between programs and maximize the cohesion of programs).

Design Specifications for Software

Deliverables:

Document the system design primarily in the form of diagrams. Supplement each diagram with text that summarizes the function (or data) and highlights important performance and design issues.

When using structured design methods, the design diagrams should:

- Depict the software as a top-down set of diagrams showing the control hierarchy of all software programs to be implemented.
- Define the function of each software program.
- Identify data and control interfaces between programs.
- Specify files, records, and global data accessed by each program.

Review Process:

Conduct structured walkthroughs to assure that the custom-built routines and programs are correctly designed.

Design Physical Model and Database Structure

Responsibility:

Project Team

Description:

The physical model is a description of the dynamics, data transformation, and data storage requirements of the software product. The physical model maps the logical model created during the Functional Design Phase to a specific technical reality. Care must be taken to retain in the physical implementation all of the capabilities inherent in the logical model.

1) Define dynamics, data transformation, and data storage requirements;
2) Map the logical model to specific technical reality

The physical model frequently differs from the logical model in the following areas:

- Constraints imposed by the data base management system - The logical data model may have different implementations in the selected data base management system.
- Performance - Data redundancies, indices, and data structure changes may have to be introduced into the physical model to improve performance.
- Distributed processing - Possible network and multiple production hardware configurations may cause changes to the physical data model.

Deliverables:

Designing the data base structure converts the data requirements into a description of the master and transient files needed to implement the requirements. If the software product will include a data base, design the data base in conjunction with the following data base management features:

- Report writer and file processing capabilities
- Online query processing to retrieve data
- Automated data dictionary systems

Review Process:

Document the physical model for incorporation into the System Design Document. Review the contents of the data dictionary entries and update to complete information on data elements, entities, files, physical characteristics, and data conversion requirements. Place a copy of all physical model and data base structure records in the Project Notebook.

Schedule structured walkthroughs to verify that the physical model and data dictionary are correct and complete.

Develop Integration Test Plan

Responsibility:

Project Team Programmers

Description:

The purpose of integration testing is to verify the integrity of a module (a cohesive set of programs) and its interfaces with other modules within the software structure. An integration test plan is developed to incorporate successfully unit-tested modules into the overall software structure and to test each level of integration to isolate errors introduced by newly incorporated modules.

1) Verify the integrity of a module;
2) Develop integration test plan;
3) Address integration levels

The number of integration levels, the classes of tests to be performed, and the order in which routines and builds are incorporated into the overall software structure are addressed in the Integration Test Plan. The following factors should be considered:

- Are routines to be integrated in a pure top-down manner or should builds be developed to test sub-functions first?
- In what order should major software functions be incorporated?
- Is the scheduling of module coding and testing consistent with the order of integration?
- Is special hardware required to test certain routines?

Integration testing should include tests that validate the following functions:

- Verify each interface between the module and all other modules.
- Access each input message or command processed by the module.
- Check each external file or data record referenced by coding statements in the module.
- Output each message, display, or record generated by the module.

An important consideration during integration test planning is the amount of test software (e.g., drivers, test case generation) that must be developed to adequately test the required functionality. For example, it may be cost-effective to delay testing of a communication function until hardware is available rather than generate test software to simulate communication links.

Similarly, it may be better to include certain completed modules in the software structure in order to avoid having to develop software drivers. These decisions are made on the basis of cost and risks.

Deliverables:

Develop the draft Integration Test Plan that addresses the following activities:

- Define the integration tests at each element level, stating objectives, what is to be tested, and verified. Testing is from the point of view of structure and function.
- Define all aspects of the formal interfaces that must undergo formal Integration testing. Review interface requirements to ensure completeness, consistency, and effectiveness.

Develop Integration Test Plan

Deliverables Continued:	❑ Plan for test tools and software that must be developed to adequately test the required functionality. ***Note:*** The Integration Test Plan may be incorporated in the Project Test Plan.
Review Process:	Conduct a peer review or structured walkthrough to assure that the draft Integration Test Plan is accurate and complete. The Integration Test Plan will be reviewed and revised as needed during the Programming Phase.

Develop System Test

Responsibility:

Project Test Team

Description:

The objectives of the system test process are to assure that the software product adequately satisfies the project requirements; functions in the computer operating environment; successfully interfaces between procedures, operating procedures, and other systems; and protects the software and data from security risks. The system should be tested under the same kind of daily conditions that will be encountered during regular operations. System timing, memory, performance, and security functions are tested to verify that they perform as specified. The functional accuracy of logic and numerical calculations are tested for verification under normal and load conditions.

Test data should be varied and extensive enough to enable the verification of the operational requirements. Expected output results should be included in the test plan in the form of calculated results, screen formats, hardcopy output, pre-determined procedural results, warnings, error messages and recovery.

Detailed planning for the system testing helps to ensure that system acceptance will be successfully completed on schedule. When applicable, system testing must include the following types of tests:

- Performance tests that measure throughput, accuracy, responsiveness, and utilization under normal conditions and at the specified maximum workload.
- Stress tests to determine the loads that result in appropriate, non-recoverable, or awkward system behavior.
- Interface tests to verify that the system generates external outputs and responds to external inputs as prescribed by approved interface control documentation.
- System recovery and reconfiguration tests.
- Verification that the system can be properly used and operated in accord with its users guide and operating instructions.
- Verification that the system meets its requirements for reliability, maintainability, and availability, including fault tolerance and error recovery.
- Verification of the effectiveness of error detection and analysis, and automated diagnostic tools.
- Demonstration that the system complies with its serviceability requirements such as accessibility, logistics, upgrades, diagnostics, and repair capabilities.

Develop System Test

Deliverables:

Develop a draft System Test Plan that describes the testing effort, provides the testing schedule, and defines the complete range of test cases that will be used to assure the reliability of the software. The test cases must be complete and the expected output known before testing is started. The test plan should address the following:

- Provide a definition of and the objectives for, each test case.
- Define the test scenario(s) including the step-by-step procedure, the number of processing cycles to be tested or simulated, and the method and responsibility for feeding test data to the system.
- Define the test environment including the hardware and software environment under which the testing will be conducted. Identify and describe manual procedures, automated procedures, and test sites (real or simulated).
- Identify test tools and special test support needs (e.g., hardware and software to simulate operational conditions or test data that are recordings of live data).
- Identify responsibilities for conducting tests; for reviewing, reporting, and approving the results; and for correcting error conditions.
- Develop a requirements verification matrix mapping individual tests to specific requirements and specifying how each system requirement will be validated.
- Schedule for integrating and testing all components including adequate time for retesting.

Note: The System Test Plan may be incorporated into the Project Test Plan.

Review Process:

Conduct peer reviews or structured walkthroughs to assure that each system test procedure is accurate, complete, and accomplishes the stated objectives. The System Test Plan will be reviewed and revised as needed during the Programming Phase.

Develop Conversion Plan

Responsibility:

Project Team

Description:

If the software product will replace an existing automated system, develop a Conversion Plan. The major elements of the Conversion Plan are to develop conversion procedures, outline the installation of new and converted files/data bases, coordinate the development of file-conversion programming, and plan the implementation of the conversion procedures.

File conversion should include a confirmation of file integrity. Determine what the output in the new system should be compared with the current system, and ensure that the files are synchronized. The objective of file conversion is new files that are complete, accurate and ready to use.

Many factors influence data conversion, such as the design of the current and new systems and the processes for data input, storage, and output. Understanding the data's function in the old system and determining if the function will be the same or different in the new system is of major importance to the Conversion Plan. The structure of the data to be converted can limit the development of the system and affect the choice of software.

Deliverables:

Develop a Conversion Plan that identifies what conversions are needed and how the conversion(s) will be implemented. Consider the following factors during the development of the Conversion Plan:

- Determine if any portion of the conversion process should be performed manually.
- Determine whether parallel runs of the old and new systems will be necessary during the conversion process.
- Understand the function of the data in the old system and determine if the use will be the same or different in the new system is important.
- The order that data is processed in the two systems influences the conversion process.
- Volume considerations, such as the size of the database and the amount of data to be converted, influence how the data will be converted. Especially important are the number of reads that are necessary, and the time these conversions will take.
- Customer work and delivery schedules, timeframes for reports and end-of-year procedures, and the criticality of the data help determine when data conversion should be scheduled.
- Determine whether data availability and use should be limited during the conversion.
- Plan for the disposition of obsolete or unused data that is not converted.
- Develop a rollback plan/fallback position in case of failure.

Develop Conversion Plan

Review Process:	Conduct structured walkthroughs to assure that the Conversion Plan is accurate and complete.
Resource:	A Conversion Plan template is available on the Research and Policy Web site at: http://www.michigan.gov/dit.

Develop System Design

Responsibility:

Project Team

Description:

1) Translate requirements into precise descriptions of the software components;
2) Obtain approval of system design

The system design is the main technical deliverable of the System Design Phase. The system design translates requirements into precise descriptions of the software components, interfaces, and data necessary before coding and testing can begin. It is a blueprint for the Programming Phase, based on the software structure and data model established in the Functional Design Phase.

The system design plays a pivotal role in the development and maintenance of a software product. The design provides valuable information used by the project manager, quality assurance staff, configuration management staff, software designers, programmers, testers, and maintenance personnel.

The system design is baselined after the system owner's formal approval of the design as described in the System Design Document. Once the system design is baselined, any changes to the design must be managed under change control procedures established in the Software Configuration Management Plan. Approved changes must be incorporated into the System Design Document.

It is important for the system owner/customers to understand that some changes to the baselined system design may affect the project scope and therefore can change the project cost, resources, or schedule. It is the responsibility of the project manager and team to identify system owner/customer requested changes that would result in a change of project scope; evaluate the potential impact to the project costs, resources, or schedule; and notify the system owner of the project planning revisions that will be required to accommodate their change requests.

Deliverables:

Each requirement identified in the Software Requirements Specification must be traceable to one or more design entities. This traceability ensures that the software product will satisfy all of the requirements and will not include inappropriate or extraneous functionality. Expand the Requirements Traceability Matrix developed in the Requirements Definition Phase to relate the system design to the requirements. Place a copy of the expanded matrix in the Project Notebook. Refer to each task for other applicable deliverables.

Review Process:

Conduct a structured walkthrough of the Requirements Traceability Matrix. Refer to task *Section 7, Conduct Critical Design Review*, for the system design review process.

Tasks:

The following tasks are involved in developing the system design:

Develop System Design Document
Conduct Critical Design Review

Develop System Design Document

Description:	The System Design Document records the results of the system design process and describes how the software product will be structured to satisfy the requirements identified in the Software Requirements Specification. The System Design Document is a translation of the requirements into a description of the software structure, software components, interfaces, and data necessary to support the programming process.
Deliverables:	Prepare the System Design Document and submit it to the system owner and customers for their review and approval. The approved System Design Document is the official agreement and authorization to use the design to build the software product. Approval implies that the design is understood, complete, accurate, and ready to be used as the basis for the subsequent lifecycle phases. In other words, once approved this becomes the design baseline. Subsequent changes or additions to the software design that receive stakeholder concurrence supersede the existing baseline and establish a new design baseline. Place a copy of the approved System Design Document in the Project Notebook.
Review Process:	Conduct structured walkthroughs as needed to ensure that the System Design Document is accurate and complete. The completion of the System Design Document is an appropriate time to schedule an In-Phase Assessment (IPA).
SDLC Reference:	*Appendix D, In-Phase Assessment Process Guide* provides a description and instructions for conducting an IPA.
Resource:	A System Design Document template is available on the Research and Policy Web site at: http://www.michigan.gov/dit.

Conduct Critical Design Review

Description:

The Critical Design Review is a formal technical review of the system design. The purpose of the review is to demonstrate to the system owner and customers that the system design can be implemented on the selected platform and accounts for all software and data requirements and accommodates all design constraints (e.g., performance, interface, security, safety, resource, and reliability requirements). The design review should include a review of the validity of algorithms needed to perform critical functions.

Several short Critical Design Reviews can replace one long review if the software consists of several components that are not highly interdependent. The review process should be a series of presentations by the project team to the system owner and other approval authorities.

Conduct a Critical Design Review that demonstrates that the design specifications are capable of supporting the full functionality of the software product, as follows:

- All algorithms will perform the required functions.
- The specification is complete, unambiguous and well documented, including timing and sizing, and data and storage allocations.
- The specification is necessary and sufficient for, and directly traceable to, the software system design.
- The specification is compatible with every other specification, piece of equipment, facility, and item of system architecture, especially as regards information flow, control, and sequencing.
- The specification is consistent with the abilities of current development and customer personnel.

In addition to verifying individual specifications, the Critical Design Review assesses other project deliverables to ensure the following:

- The team is following the approved design approach.
- Measures to reduce risk on a technical, cost, and schedule basis are adequate.
- The performance characteristics of the design solution are acceptable.
- Testing will be sufficient to ensure software product correctness.
- The resultant application will be maintainable.
- Provisions for automatic, semi-automatic, and manual recovery from hardware/software failures and malfunctions are adequate and documented.

Conduct Critical Design Review

Description Continued:

- Diagnostic programs, support equipment, and commercial manuals all comply with the system maintenance concept and specification requirements.

Deliverables:

Create and distribute official meeting minutes for each design review session. The minutes should consist of significant questions and answers, action items and individual/group responsible, deviations, conclusions, and recommended courses of action resulting from presentations or discussions. Recommendations that are not accepted should be recorded along with the reason for non-acceptance. Minutes must be distributed to review participants. The system owner determines review performance as follows:

- **Approval** - The review was satisfactorily completed.
- **Contingent Approval** - The review is not finished until the satisfactory completion of resultant action items.
- **Disapproval** - The specification is inadequate. Another Critical Design Review will be required.

Review Process:

Not applicable.

Develop Program Specifications

Responsibility:

Project Team

Description:

A Program Specification is a written procedural description of each software system routine. The Program Specification should provide precise information needed by the programmers to develop the code.

Many techniques are available for specifying the system design, such as formal specification languages, program design languages (e.g., pseudo-code or structured English), meta-code, tabular tools (e.g., decision tables), and graphical methods (e.g., flow charts or box diagrams). In object-oriented design, the specification of requirements and preliminary design constraints and dependencies often results in the design language producing the detailed specifications.

Select the technique or combination of techniques that is best suited to the software project and to the experience and needs of the programmers who will use the system design as their blueprint. The following are suggestions for using the techniques:

- Decision trees are useful for logic verification or moderately complex decisions that result in up to 10-15 actions. Decision trees are also useful for presenting the logic of a decision table to customers.
- Decision tables are best used for problems involving complex combinations of up to 5-6 conditions. Decision tables can handle any number of actions; however, large numbers of combinations of conditions can make decision tables unwieldy.
- Structured English is best used wherever the problem involves combining sequences of actions with decisions or loops. Once the main work of physical design has been done and physical files have been defined, it becomes extremely convenient to be able to specify physical program logic using the conventions of structured English, but without getting into the detailed syntax of any particular programming language (pseudo-code).
- Standard English is best used for presenting moderately complex logic once the analyst is sure that no ambiguities can arise.

Deliverables:

Specifications may be developed as documents, graphic representations, formal design languages, records in a data base management system, and CASE tool dictionaries. A list of program characteristics typically included in a Program Specification is provided at the end of this section.

Review Process:

Conduct a series of structured walkthroughs to ensure that the Program Specification is accurate and complete.

Develop Program Specifications

Sample Characteristics:

For each program to be custom-built, define the program's functional and technical characteristics, as they become known. The following is a list of program characteristics:

- ❑ Program identification
- ❑ Program name
- ❑ Program generic type
- ❑ Functional narrative
- ❑ Program hierarchical features diagram
- ❑ Development dependencies and schedule
- ❑ Operating environment
  - equipment
  - programming language and version
  - preprocessor
  - operating system
  - storage restrictions
  - security
- ❑ Frequency of run
- ❑ Data volumes
- ❑ Program termination messages
  - normal termination
  - abnormal termination
- ❑ Console/printer messages
- ❑ Recovery/restart procedures
- ❑ Software objectives
- ❑ Program input/output diagram
- ❑ Data bank information
- ❑ Called and calling programs/modules
- ❑ Program logic diagrams
- ❑ Significant "how-to" instructions
- ❑ Telecommunications information

Define Programming Standards

Responsibility:

Project Team Programmers

Description:

Programming standards are necessary to ensure that custom-built software has acceptable design and structural properties. Programming standards must be practical, easy to implement, and accepted by the project team. The project team programmers should be the primary developers of the standard. Use a structured approach to programming to allow for easy modification and to facilitate testing and debugging.

The following guidelines are generally applicable to any programming language. Use these guidelines as the basis for the programming standard and add project-specific standards relating to the programming language and tools:

- Control Flow Constructs
  - sequence
  - if-then-else
  - case statement
  - do-while (pretest loop)
  - do-until (post-test loop)
- Module Size
  - Number of executable lines of source code should average 100 lines per unit.
  - Units should contain no more than 200 lines of executable source code.
- Module Design
  - Units do not share temporary storage locations for variables
  - Units perform a single function
  - Avoid self-modifying code
  - Each unit is uniquely named
  - Each unit has a standard format:
    - prologue
    - variable declarations
    - executable statements/comments
  - Use single entry/exit points except for error paths
  - Set comments off from the source code in a uniform manner
- Symbolic Parameters
  - Use instead of specific numerics
  - Use for constants, size of data structures, relative position in list
- Naming Conventions
  - Use uniform naming throughout each unit and module to be put under configuration control
  - Use meaningful variable names
  - Do not use keywords as identifiers
- Mixed Mode Operations
  - Avoid mixed mode expressions
  - Add comments in code whenever used

Define Programming Standards

Description Continued:

- Error and Diagnostic Messages
  - Design messages to be self-explanatory and uniform
  - Do not require customer to perform table lookups
- Style
  - Use conventions such as indentation, white space, and blank lines to enhance readability
  - Align compound statements
  - Avoid "go to" statements.
  - Avoid compound, negative Boolean expressions
  - Avoid nesting constructs beyond five levels deep
  - Avoid deeply nested "if" statements.
  - Use parentheses to avoid ambiguity
  - Include only one executable statement per line
  - Avoid slick programming tricks that may create or encourage defects or be difficult to maintain; the most direct solution is best.

Deliverables:

Create a programming standards document and distribute the document to all project team members. An existing programming standard can be used if it is applicable to the programming language and tools being used for the project.

Review Process:

Conduct a peer review to assure that the programming standards are complete and appropriate for the project's programming language and tools.

Revise Project Plan

Responsibility:	Project Manager
Description:	Once the Critical Design Review is completed, the system design is baselined, and the deliverables from the Functional Design Phase have been updated as needed to reflect changes caused by the system design, determine if the project estimates for resources, cost, and schedule need to be revised.
Deliverables:	Review the Project Plan for accuracy and completeness of all System Design Phase activities and make any changes needed to update the information. Expand the information for the Programming Phase to reflect accurate estimates of resources, costs, and hours. Place a copy of the revised Project Plan in the Project Notebook. ***Note:*** A Project Plan is an effective management tool that is recommended for all projects regardless of size. The plan can be consolidated for small projects.
Review Process:	Conduct a structured walkthrough to ensure that the Project Plan reflects the project's current status and adequately estimates the resources, costs, and schedule for the Programming Phase. The Project Plan is formally reviewed during the In-Phase Assessment and Phase Exit processes.

Conduct System Design Exit

Conduct Structured Walkthroughs

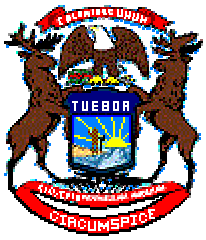
Responsibility:	Project Manager, Work Product Author and Reviewers
Description:	During the System Design Phase, schedule at least one structured walkthrough to review each of the System Design Phase deliverables, i.e., Physical Model, draft of the Integration Test Plan, draft of the System Test Plan, Conversion Plan, System Design Document, Program Specifications, and Programming Standards. ***Note:*** A structured walkthrough is an effective project management tool that is recommended for all projects regardless of size; however, if an item has gone through a prior formal structured walkthrough, and the modifications are minor, a less formal review may be warranted.
SDLC Reference:	Appendix C, *Conducting Structured Walkthroughs*, provides a procedure and sample forms that can be used for structured walkthroughs.

Conduct In-Phase Assessment

Responsibility:	Project Manager and Independent Reviewer
Description:	Schedule at least one IPA prior to the System Design Phase Exit process. Additional IPAs can be performed during the phase, as appropriate. The completion of System Design Document is an appropriate time to schedule an IPA. ***Note:*** An IPA is an effective project management tool that is recommended for all projects regardless of size.
SDLC Reference:	*Appendix D, In-Phase Assessment Process Guide*, provides a procedure and sample report form that can be used for In-Phase Assessments.

Conduct System Design Phase Exit

Responsibility:	Project Manager
Description:	Schedule the Phase Exit as the last activity of the System Design Phase. It is the responsibility of the project manager to notify the appropriate participants when a project is ready for the Phase Exit process and to schedule the Phase Exit meeting. All functional areas and the Quality Assurance representative involved with the project should receive copies of the deliverables produced in this phase. ***Note:*** A Phase Exit is an effective project management tool that is recommended for all software projects regardless of size. For small software projects, phases can be combined and addressed during one Phase Exit.
SDLC Reference:	*Appendix E, Phase Exit Process Guide*, provides a procedure and sample report form that can be used for phase exits.



SYSTEMS DEVELOPMENT LIFECYCLE

SECTION 8

PROGRAMMING PHASE

Section 8: Programming Phase

Table of Contents

Section 8: Programming Phase

Highlights of Phase

Programming Phase

Inputs:

- Project Notebook
- Design specifications
- Physical model
- Data Dictionary
- Integration Test Plan *(draft)*
- System Test Plan *(draft)*
- Conversion Plan
- Requirements Traceability Matrix *(expanded)*
- System Design Document
- Program Specifications
- Programming Standards
- Project Plan *(revised)*
- Software Quality Assurance Plan

Outputs:

- Production Platform Acquisition Plan
- Installation Plan *(draft)*
- Software units and modules
- Requirements Traceability Matrix *(expanded)*
- Integration Test Plan *(final)*
- System Test Plan *(final)*
- Project Test File
- Development baselines
- Transition Plan
- Operating Documentation *(draft)*
  - Procedure Manual
  - Programmers Reference Manual
  - Administrators Manual
  - Operations Manual
- Training Plan *(draft)*
- Project Plan *(revised)*

Key Activities:

Develop Production Platform Acquisition Plan
Develop Production Platform Installation Plan
Establish Programming Environment
Write Programs
Conduct Unit Testing
Establish Development Baselines
Plan Transition to Operational Status
Generate Operating Documentation
Develop Training Program
Revise Project Plan
Conduct Structured Walkthrough(s)
Conduct In-Phase Assessment
Conduct Programming Phase Exit

Forms Utilized

Production Platform Acquisition Plan
Installation Plan
Transition Plan
Training Plan

Methods and Tools

Software:

- purchase off the shelf software
- develop custom software
- decide on platforms (operating systems)
- user support issues considered

Hardware:

- hardware systems, vendor selection
- connectivity allowing for future growth
- support offered by vendors

Train users:

- consider users and operators
- use internal or external training?
- Training tools
  - manuals
  - video, multimedia presentations
  - tutorials
- Get feedback on system

Testing of system:

- ensure quality and reliability
- use live sample data if possible
- user acceptance and Test procedures

Section 8: Programming Phase

Overview

Description:

In this phase any hardware or software procured to support the programming effort is installed. Plans are developed for the acquisition and installation of the operating environment hardware and software. A training program is designed and a Training Plan that describes the program is developed.

1) *Procured hardware and/or software is installed;*
2) *Develop plan to acquire and installation of operating environment hardware and software;*
3) *Develop a training plan;*
4) *Source code is generated;*
5) *Database utilities are coded;*
6) *Object code compiled;*
7) *Unit testing performed;*
8) *Operating documentation is developed*

The activities in this phase result in the transformation of the system design into the first complete representation of the software product. The source code, including suitable comments, is generated using the approved program specifications. If the software product requires a database, any data base utilities are coded. The source code is then grouped into processable units and all high-level language units are compiled into object code. Unit testing is performed to determine if the code satisfies the program specifications and is complete, logical, and error free.

The operating documentation is also developed. The operating documentation is required for installing, operating, and supporting the software product through its lifecycle.

Review Process:

Quality reviews are necessary during this phase to validate the product and associated deliverables. The activities that are appropriate for quality reviews are identified in this section and the Lifecycle Model section. The time and resources needed to conduct the quality reviews should be reflected in the project resources, schedule, and work breakdown structure.

SDLC References:

Section 2 Lifecycle Model, Quality Reviews, provides an overview of the Quality Reviews to be conducted on a project.

Appendix C, Conducting Structured Walkthroughs, provides a procedure and sample forms that can be used for structured walkthroughs.

Appendix D, In-Phase Assessment Process Guide, provides a procedure and sample report form that can be used for in-phase assessments.

Appendix E, Phase Exit Process Guide, provides a procedure and sample report form that can be used for phase exits.

Develop Production Platform Acquisition Plan

Responsibility:	Project Team
Description: *1) Develop a plan for acquisition of hardware and software, and communications equipment;* *2) Develop a contingency plan*	Develop a plan for the acquisition of any hardware, software, and communications equipment needed to install and operate the software product at all system owner and customer sites. The plan should address any special procurements necessary to accommodate the hardware and communications equipment that may exist at a particular site. Acquisition planning must include sufficient lead-time to accomplish all procurement, delivery, testing, and installation processes. It may be necessary to perform a risk analysis of the impact of certain resources not being available when needed. Develop a contingency plan for dealing with needed resources that are acquired later than expected.
Deliverables:	Work closely with the system owner and representatives from the customer sites to assure that all site-specific hardware, software, and communications needs are addressed in the Production Platform Acquisition Plan. Place a copy of the Production Platform Acquisition Plan in the Project Notebook. ***Note:*** For projects that do not require extensive procurement and installation of hardware and software, the Production Platform Acquisition and Installation Plans can be combined into one deliverable.
Review Process:	Conduct a structured walkthrough to assure that the Production Platform Acquisition Plan is accurate and complete.
Resource:	A Production Platform Acquisition Plan template is available on the Research and Policy Web site at: http://www.michigan.gov/dit.

Develop Installation Plan

Responsibility:

Project Team

Description:

1) Prepare installation plan

The Installation Plan is prepared to specify the requirements and procedures for the full-scale installation of the developed software product at the system owner's and all customers' work sites. The plan also addresses the installation of any hardware, off-the-shelf software, firmware, and communications equipment needed to operate the product at each site. In developing an Installation Plan consider each site's requirements for continuity of operations, level of service, and the needs of the project team, customers, maintenance personnel, and management.

Deliverables:

Work closely with the system owner and representatives from the customer sites to assure that all site-specific hardware, software, and communications installation requirements are addressed in the Installation Plan. Develop a draft Installation Plan that addresses the following issues:

- ❑ Schedule of all installation activities.
- ❑ Items to be delivered to each installation site.
- ❑ Number and qualifications of personnel performing installation.
- ❑ Equipment environmental needs and installation instructions.
- ❑ Hardware, software, firmware, tools, documentation, and space required for each installation.
- ❑ Special requirements governing the movement of equipment to each site.
- ❑ Communications requirements.
- ❑ Dependencies among activities affected by installation.
- ❑ Installation tests to assure the integrity and quality of the installed product.

Place a copy of the draft Installation Plan in the Project Notebook.

Note: For projects with limited procurement and installation requirements, the Acquisition and Installation Plans can be combined into one deliverable.

Review Process:

Conduct a structured walkthrough to assure that the draft Installation Plan is accurate and complete. The Installation Plan will be reviewed and revised as needed during the Software Integration and Testing Phase.

Resources:

An Installation Plan template is available on the Research and Policy Web site at: http://www.michigan.gov/dit.

Establish Programming Environment

Responsibility:

Project Team

Description:

Establish Programming Environment:

1) *Assemble and install hardware, software, communications equipment, databases etc.;*
2) *Conduct testing to verify that everything is operating;*
3) *Activate security procedures*

Establishing the programming environment involves assembling and installing the hardware, software, communications equipment, databases, and other items required to support the programming effort. When the installation of the equipment or software is complete, conduct testing to verify the operating characteristics and functionality of the hardware and software. If required, security software and procedures should be activated when the installations are completed.

If the operational environment is also the programming environment, it may be necessary to alter the operational environment to accommodate an infrastructure of purchased hardware and software for use during programming and testing.

Before being integrated into, or used to support, the software product, vendor products should be tested to verify that the product satisfies the following objectives:

- The product performs as advertised/specified.
- The product's performance is acceptable and predictable in the target environment (e.g., testing for LAN certification).
- The product fully or partially satisfies the project requirements.
- The product is compatible with the project team's other hardware and software tools.

Time should be planned for the project team to become familiar with new products. Ensure that the project team members who will use the hardware or software obtain proper training. This may involve attendance at formal training sessions conducted by the vendor or the services of a consultant to provide in-house training.

This is a good time to review the programming standards that were established in the System Design Phase. Make any changes to the standards that are needed to accommodate the procured hardware and software.

Deliverables:

Not applicable

Review Process:

Not applicable

Write Programs

Responsibility:

Project Team Programmers

Description:

1) Write programs;
2) Generate source and object code

This activity involves generating the source and object code for the software product. The code should be written in accordance with the programming standards developed in the System Design Phase. Regardless of the platform, development of the code should adhere to a consistent set of programming techniques and error prevention procedures. This will promote reliable, maintainable code, developed in the most efficient and cost effective manner.

The source and object code should be uniquely identified and stored in a way to facilitate the configuration control measures described in the Software Configuration Management Plan.

Writing programs includes the following tasks:

- Use the Program Specifications developed in the System Design Phase as the basis for the coding effort.
- Generate source code and machine-readable modules.
- Generate the physical files and data base structure.
- Generate video screens, report generation codes, and plotting instructions.
- If conversion of an existing system or data is necessary, generate the program(s) described in the Conversion Plan.
- Conduct preliminary testing of completed units. When the test output is correct, review the program specification to assure that the unit or module conforms to the specification.

Coding Practices:

The following coding practices should be implemented:

- The programming staff should meet at scheduled intervals to discuss problems encountered and to facilitate program integration and uniformity.
- Program uniformity should be achieved by using a standardized set of naming conventions for programs, data elements, variables, and files.
- Modules that can be shared by programs requiring the same functionality should be implemented to facilitate development and maintenance.
- Meaningful internal documentation should be included in each program.
- All code should be backed up on a daily basis and stored in an offsite location to avoid catastrophic loss.

Write Programs

Coding Practices Continued:

- A standard format for storing and reporting elements representing numeric data, dates, times, and information shared by programs should be determined.
- The System Design Document should be updated to reflect any required deviations from the documented design.

Deliverables:

The following deliverables are developed:

- Completed units and modules of code.
- Test materials generated from preliminary testing.

Each requirement identified in the Software Requirements Specification must be traceable to the code. This traceability ensures that the software product will satisfy all of the requirements and will not include inappropriate or extraneous functionality. Expand the Requirements Traceability Matrix developed in the Requirements Definition Phase to relate the source and object code to the requirements. Place a copy of the expanded matrix in the Project Notebook.

Weekly informal reviews of each programmer's work are recommended to keep the project team informed of progress and to facilitate the resolution of any problems that may occur. The combined knowledge and skills of the team members will help to build quality into the software product and support the early detection of errors in design, logic, or code.

Review Process:

Conduct structured walkthroughs on the expanded Requirements Traceability Matrix and completed units and modules to assure that the code is accurate, logical, internally well documented, complete, and error free. Structured walkthroughs should also be used to validate that the code is reliable and satisfies the program specifications and project requirements.

For large or complex projects, conduct code inspections at successive phases of code production. Code inspection is a static analysis technique that relies on visual examination of code to detect errors, violations of development standards, and other problems. These inspections are particularly important when several programmers or different programming teams are developing code. The inspection team may include experts outside of the project. Ideal times for code inspections occur when code and unit tests are complete, and when the first integration tests are complete. Code inspections should be identified as milestones in the Project Plan.

Conduct Unit Testing

Responsibility:

Project Team Programmers

Description:

Unit testing is used to verify the input and output for each module. Successful testing indicates the validity of the function or sub-function performed by the module and shows traceability to the design. During unit testing, each module is tested individually and the module interface is verified for consistency with the design specification. All-important processing paths through the module are tested for expected results. All error-handling paths are also tested.

Unit testing is driven by test cases and test data that are designed to verify software requirements, and to exercise all program functions, edits, in-bound and out-of-bound values, and error conditions identified in the program specifications. If timing is an important characteristic of the module, tests should be generated that measure time critical paths in average and worst-case situations.

Plan and document the inputs and expected outputs for all test cases in advance of the tests. Log all test results. Analyze and correct all errors and retest the unit using the scenarios defined in the test cases. Repeat testing until all errors have been corrected.

While unit testing is generally considered the responsibility of the programmer, the project manager or lead programmer should be aware of the unit test results.

Deliverables:

Completion of unit testing for a software component signifies internal project delivery of a component or module for integration with other components. Place all components that have completed unit testing under configuration control as described in the Software Configuration Management Plan. These components form the Production Baseline. Configuration controls restrict changes to tested and approved software in the Production Baseline. Subsequent changes or additions to the software that are agreed upon in a Critical Design Review and receive stakeholder concurrence supersede the existing baseline and establish a new Production Baseline.

Review the draft versions of the Integration and System Test Plans developed during the System Design Phase. Update the plans, as needed, to reflect any changes made to the software design. Deliver the final versions of the Integration and System Test Plans to the system owner and customer for review and approval. Place a copy of the approved plans in the Project Notebook.

Create a Project Test File for all test materials generated throughout the project lifecycle. Place all unit test materials (e.g., inputs, outputs, results and error logs) in the Project Test File. The test cases used for unit testing may become a subset of tests for integration testing.

Review Process:

Conduct peer reviews on the test materials to be placed in the Project Test File. Conduct structured walkthroughs on any updated plans, e.g., Integration and System Test Plans.

Establish Development Baselines

Responsibility:

Project Team Programmers

Description:

A development baseline is an approved "build" of the software product. A build can be a single component or a combination of software components. The first development baseline is established after the first build is completed, tested, and approved by the project manager or lead programmer. Subsequent versions of a development baseline should also be approved. The approved development baseline for one build supersedes that for its predecessor build.

Conduct internal build tests such as regression, functional, and performance/reliability. Regression tests are designed to verify that capabilities in earlier builds continue to work correctly in subsequent builds. Functional tests focus on verifying that the build meets its functional and data requirements and correctly generates each expected display and report. Performance and reliability tests are used to identify the performance and reliability thresholds of each build.

Once the first development baseline is established, any changes to the baseline must be managed under the change control procedures described in the Software Configuration Management Plan. Approved changes to a development baseline must be incorporated into the next build of the software product and revisions made to the affected deliverables (e.g., Software Requirements Specification, System Design Document, and Program Specifications).

Deliverables:

Document the internal build test procedures and results. Identify errors and describe the corrective action that was taken. Place a copy of the internal build test materials in the Project Test File.

Maintain configuration control logs and records as required in the Software Configuration Management Plan.

Expand the Requirements Traceability Matrix developed in the Requirements Definition Phase. All deliverables produced during the code, unit testing, and build processes must be traced back to the project requirements and system design. This traceability ensures that the product will satisfy all of the requirements and remain within the project scope. Place a copy of the expanded Requirements Traceability Matrix in the Project Notebook.

Review Process:

Conduct peer reviews on the internal build test materials to be placed in the Project Test File. Conduct structured walkthroughs on any updated documents, e.g., the Requirements Traceability Matrix.

Plan Transition to Operational Status

Responsibility: Project Team

Description: Successful transition from acceptance testing to full operational use of the software product depends on planning the transition long before the software product is installed in its operational environment. In planning for the transition, quantify the operational needs associated with the software product and describe the procedures that will be used to perform the transition. Rely on experience and data gathered from previous, similar projects to define these needs.

Deliverables: Develop a Transition Plan that describes the detailed plans, procedures, and schedules that will guide the transition process. Coordinate development of the plan with the operational and maintenance personnel. The following issues should be considered in the preparation of a Transition Plan:

- Develop detailed operational scenarios to describe the functions to be performed by the operational support staff, maintenance staff, and customers.
- Define the number and qualifications of operations and maintenance personnel and specify when they must be in place. Estimate training requirements for these people.
- Document the release process. If development is incremental, define the particular process, schedule, and acceptance criteria for each release.
- Describe the development or migration of data, including the transfer or reconstruction of historic data. Schedule ample time for the system owner and customer to review the content of reconstructed or migrated data files to reduce the chance of errors or omissions.
- Specify problem identification and resolution procedures for the operational software product.
- Define the configuration management procedures that will be used for the operational software product. Ideally, the methods defined in the Software Configuration Management Plan that were employed during product development can continue to be used for the operational product.
- Define the scope and nature of support that will be provided by the project team during the transition period.
- Specify the organizations and individuals that will be responsible for each transition activity, ensuring that responsibility for the software product by the operations and maintenance personnel increases progressively.
- Identify products and support services that will be needed for day-to-day operations or that will enhance operational effectiveness.

Plan Transition to Operational Status

Review Process:	Conduct a structured walkthrough to assure that the Transition Plan is logical, accurate, and complete. Involve operational and maintenance personnel in the walkthrough.
Resources:	A Transition Plan template is available on the Research and Policy Web site at: http://www.michigan.gov/dit.

Generate Operating Documentation

Responsibility:

Project Team/Technical Writer

Description:

Plan, organize, and write the operating documentation that describes the functions and features of the software product from the customers point-of-view. The different ways that customers (including system administration and maintenance personnel) will interact with the software product must be considered. The needs of the customers should dictate the document presentation style and level of detail. Responsibilities for changing and maintaining the documents should be described in each document.

The following are typical operating documents for a large software project:

- Procedure Manual
- Programmers Reference Manual
- Systems Administration Manual
- Data Base Administration Manual
- Operations Manual

It is recommended that a technical writer be involved in the generation of all operating documents. A technical writer works closely with the project team to ensure that documents are grammatically correct; comply with applicable standards; and are consistent, readable, and logical.

Note: The operating documents can be developed as separate manuals or combined to accommodate less complex software projects.

Steps:

Use the following steps to develop the operating documentation:

- Identify the operating documents that need to be developed. Determine if any of the documents can be combined or delivered as multiple volumes.

- Determine whether the documents should be provided as printed material, standalone electronic files, online documentation accessed through the software product, or a combination.

- Determine the best presentation method or combination of methods required for each of the documents, such as a traditional manual, quick reference guide or card, or online help.

- Identify all of the features of the software customer interface and the tasks customers will perform.

- Identify the customers' needs and experience levels to determine:
  - The amount of customer interaction, level of interaction, and whether the interaction is direct or indirect.
  - The appropriate level of detail (e.g., the Procedure Manual should not contain highly technical terms and explanations that may confuse or frustrate a customer).

Generate Operating Documentation

Steps Continued:	❑ Determine the document content and organization based on whether the document will be used more as an instructional tool or a reference guide. ❑ Develop descriptions of each function and feature of the software product and organize the information to facilitate quick, random access. Provide appropriate illustrations to enhance clarity and understanding. ❑ Establish a schedule for the documents to be reviewed after the software product goes into production. Operating documents must be kept up-to-date as long as the software product remains in production.
Deliverables:	Refer to each task for applicable deliverables.
Review Process:	Refer to each task for applicable review processes.
Tasks:	The following tasks describe the minimum requirements for operating documentation: Develop Procedures Manual Develop Programmers Reference Manual

Develop Procedures Manual

Description:

The Procedure Manual provides detailed information customers need to access, navigate through, and operate the software product. Customers rely on the Procedure Manual to learn about the software or to refresh their memory about specific functions. A Procedure Manual that is organized functionally so that the information is presented the same way the software product works helps customers understand the flow of menus and options to reach the desired functions.

Different categories of customers may require different types of information. A modular approach to developing the Procedure Manual to accommodate the needs of different types of customers eliminates duplication and minimizes the potential for error or omission during an amendment or update. For example, separate general information that applies to all customers from the special information that applies to selected customers such as system administrators or data base administrators. The special information can be presented in appendixes or supplements that are only provided to the customers who need the information.

Deliverables:

Write the draft Procedure Manual in clear, non-technical terminology that is oriented to the experience levels and needs of the customer(s). The following are typical features of a procedure manual:

- Overview information on the history and background of the project and the architecture, operating environment, and current version or release of the software product.
- Instructions for how to install, setup, or access the software product. Complete coverage of all software functions, presented in a logical, hierarchical order.
- Accurate pictures of screens and reports, ideally with data values shown, so the customer can easily relate to examples.
- In-depth examples and explanations of the areas of the software product that are most difficult to understand.
- Clear delineation of which features are accessible only to specific customers.
- Instructions on accessing and using online help features.
- Procedures for data entry.
- Descriptions of error conditions, explanations of error messages, and instructions for correcting problems and returning to the function being performed when the error occurred.
- Instructions for performing queries and generating reports.
- Who to contact for help or further information.

Develop Procedures Manual

Deliverables Continued:

Note: For large or complex software products, separate manuals (e.g., Procedure Manual, Data Base Administrator's Manual, and System Administrator's Manual) may be necessary to address the needs of different categories of customers.

For very small projects, a quick reference guide or card may be more appropriate than a full-scale Procedure Manual. The guide or card should be designed to provide a quick reference of logon, logoff, and commands for frequently used functions.

For projects of any size, a quick reference card may be developed as a supplement to more detailed customer documentation.

Review Process:

Conduct structured walkthroughs for the draft Procedure Manual or set of customer documents to assure that the documentation is complete, easy to use, and accurately reflects the software product and its functions.

The draft customer documentation will be tested and verified with the software product during the Software Integration and Testing Phase.

Develop Programmers Reference Manual

Description:

The Programmers Reference Manual contains programming information used by the maintenance staff to maintain the programs, databases, interfaces, and operating environment. The Programmers Reference Manual should provide an overall conceptual understanding of how the software product is constructed and the details necessary to implement corrections, changes, or enhancements.

The Programmers Reference Manual describes the logic used in developing the software product and the functional and system flow to help the maintenance programmers understand how the programs fit together. The information should enable a programmer to determine which programs may need to be modified to change a system function or to fix an error.

Deliverables:

The following are typical features of a Programmers Reference Manual:

- A description of the technical environment, including versions of the programming language(s) and other proprietary software packages.
- A brief description of the design features including descriptions of unusual conditions and constraints.
- An overview of the software architecture, program structure, and program calling hierarchy.
- The design and programming standards and techniques used to develop the software product.
- Concise descriptions of the purpose and approach used for each program.
- Layouts for all data structures and files used in the software product.
- Descriptions of maintenance procedures, including configuration management, program checkout, and system build routines.
- The instructions necessary to compile, link, edit, and execute all programs.
- Manual and automated backup procedures.
- Error processing features.

Use appendixes to provide detailed information that is likely to change as the software product is maintained. For example, a list of program names and a synopsis of each program could be included as an appendix.

Review Process:

Conduct structured walkthroughs of the draft Programmers Reference Manual to assure that the documentation is complete, easy to use, and accurately reflects the software product and its functions.

The draft Programmers Reference Manual will be tested and verified with the software product during the Software Integration and Testing Phase.

Develop Training Program

Responsibility:

Project Team

Description:

A Training Program defines the training needed to implement and operate the software product successfully. The Training Plan should address the training that will be provided to the system owner, customers, and maintenance staff. When new hardware or software is being used, affected personnel will need hands-on experience before bringing the new equipment or software into daily operation.

Training must address both the knowledge and the skills required operating and using the system effectively. Design the training program to accomplish the following objectives:

- Provide trainees with the specific knowledge and skills necessary to perform their work.
- Prepare training materials that will sell the software product as well as instruct the trainees. The training program should leave the trainees with the enthusiasm and desire to use the new product.
- Account for the knowledge and skills the trainees bring with them, and use this information as a transition to learning new material.
- Anticipate the needs for follow-on training after the software product is fully operational, including refresher courses, advanced training, and repeats of basic courses for new personnel.
- Build in the capability to update the training as the software product evolves.

Involve the system owner and key customers in the planning to determine the education and training needs for all categories of customers (managers, customers, and maintenance staff).

Deliverables:

Prepare a draft Training Plan that describes the Training Program and at a minimum addresses the following issues:

- Identifies personnel to be trained. Review the list of trainees with the system owner and customers to ensure that all personnel who should receive training have been identified.
- Defines the overall approach to training and the required training courses.
- Establishes the scope of the training needed for customers, management, operations, and maintenance personnel.
- Define how and when training will be conducted. Specify instructor qualifications, learning objectives, and mastery or certification requirements (if applicable).

Develop Training Program

Deliverables Continued:

- ❑ Identify any skill areas for which certification is necessary or desirable. Tailor the training to the certification requirements.
- ❑ Establish a preliminary schedule for the training courses. The schedule must reflect training requirements and constraints outside the project. Schedule individual courses to accommodate personnel who may require training in more than one area. Identify critical paths in the training schedule such as the time period for the software product's installation and conversion to production status.
- ❑ Define the required course(s), outline their content and sequence, and establish training milestones to meet transition schedules.
- ❑ Tailor the instruction methods to the type of material being presented. Include classroom presentation, interactive computer-assisted instruction, demonstrations, individual video presentations, and hands-on experience, either live or simulated.
- ❑ Identify trainers who are technically knowledgeable and were involved in the design and development of the system. For projects with extensive and formal training requirements, it may be necessary to provide training for the trainers.
- ❑ Consider availability of the following: customers, system-tested software, training rooms and equipment, and the completion of system documentation and training materials.

You may also include the following:

- ❑ Identify the organization's training policy for meeting training needs.
- ❑ Ensure software managers have received orientation on the training program
- ❑ Ensure training courses prepared at the organization level are developed and maintained according to organizational standards.
- ❑ Ensure a wavier procedure for required training is established and used to determine whether individuals already possess the knowledge and skills required to perform in their designated area.
- ❑ Ensure measurements are made and used to determine the status of the training program activities.
- ❑ Ensure training program activities are reviewed with senior management on a periodic basis.
- ❑ Ensure the training program is independently evaluated on a periodic basis for consistency with, and relevance to, the organization's needs.

Develop Training Program

Deliverables Continued:	❑ Ensure the training program activities and deliverables are reviewed and/or audited and the results are reported. ❑ Ensure training records are properly maintained. Place a copy of the draft Training Plan in the Project Notebook. The plan will be reviewed and updated during the Software Integration and Testing Phase.
Review Process:	Conduct a structured walkthrough to assure that the draft Training Plan is accurate and complete.
Resource:	A Training Plan template is available on the Research and Policy Web site at: http://www.michigan.gov/dit

Conduct Programming Phase Exit

Responsibility:	Project Manager
Description:	Once the coding effort is completed and unit and integration testing have been conducted, determine if the project estimates for resources, cost, and schedule need to be revised.
Deliverables:	Review the Project Plan for accuracy and completeness of all Programming Phase activities and make any changes needed to update the information. Expand the information for the Software Integration and Testing Phase to reflect accurate estimates of resources, costs, and hours. Place a copy of the revised Project Plan in the Project Notebook. ***Note:*** A Project Plan is an effective management tool that is recommended for all projects regardless of size. The plan can be consolidated for small projects.
Review Process:	Conduct a structured walkthrough to ensure that the Project Plan reflects the project's current status and adequately estimates the resources, costs, and schedule for the Software Integration and Testing Phase. The Project Plan is formally reviewed during the In-Phase Assessment and Phase Exit processes.

Conduct Programming Phase Exit

Conduct Structured Walkthroughs

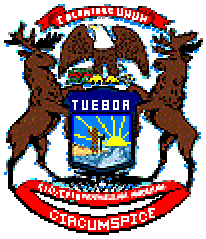
Responsibility:	Project Manager, Work Product Author and Reviewers
Description:	During the Programming Phase, schedule at least one structured walkthrough to review each of the Programming Phase deliverables, i.e., Acquisition Plan, draft of the Installation Plan, Integration Test Plan, System Test Plan, Software Baseline, Transition Plan, draft of the Operating Documents, and the draft Training Plan. ***Note:*** A structured walkthrough is an effective project management tool that is recommended for all projects regardless of size; however, if an item has gone through a prior formal structured walkthrough, and the modifications are minor, a less formal review may be warranted.
SDLC Reference:	*Appendix C, Conducting Structured Walkthroughs*, provides a procedure and sample forms that can be used for structured walkthroughs.

Conduct In-Phase Assessment

Responsibility:	Project Manager and Independent Reviewer
Description:	Schedule at least one IPA prior to the Programming Phase Exit process. Additional IPAs can be performed during the phase, as appropriate. ***Note:*** An IPA is an effective project management tool that is recommended for all projects regardless of size.
SDLC Reference:	*Appendix D, In-Phase Assessment Process Guide*, provides a procedure and sample report form that can be used for In-Phase Assessments.

Conduct Programming Phase Exit

Responsibility:	Project Manager
Description:	Schedule the Phase Exit as the last activity of the Programming Phase. It is the responsibility of the project manager to notify the appropriate participants when a project is ready for the Phase Exit process and to schedule the Phase Exit meeting. All functional areas and the Quality Assurance representative involved with the project should receive copies of the deliverables produced in this phase. ***Note:*** A Phase Exit is an effective project management tool that is recommended for all software projects regardless of size. For small software projects, phases can be combined and addressed during one Phase Exit.
SDLC Reference:	*Appendix E, Phase Exit Process Guide*, provides a procedure and sample report form that can be used for phase exits.



SYSTEMS DEVELOPMENT LIFECYCLE

SECTION 9

SOFTWARE INTEGRATION AND TESTING PHASE

Section 9: Integration and Testing Phase

Table of Contents

Section 9: Integration and Testing Phase

Highlights of Phase

Software Integration and Testing Phase

Inputs:

- ❑ Project Notebook
- ❑ Acceptance Test Plan *(draft)*
- ❑ Acquisition Plan
- ❑ Installation Plan *(draft)*
- ❑ Software modules
- ❑ Requirements Traceability Matrix *(expanded)*
- ❑ Project Test File
- ❑ Development baselines
- ❑ Transition Plan
- ❑ Operating Documentation *(draft)*
  - Procedure Manual
  - Programmers Reference Manual
- ❑ Training Plan *(draft)*
- ❑ Integration Test Plan
- ❑ System Test Plan
- ❑ Project Plan
- ❑ Software Quality Assurance Plan

Outputs:

- ❑ Integration Test Reports
- ❑ System Test Report
- ❑ Operating Documents *(final)*
  - Procedure Manual
  - Programmers Reference Manual
- ❑ Training Plan *(final)*
- ❑ Installation Plan *(final)*
- ❑ Acceptance Test Report
- ❑ Acceptance Checklist
- ❑ Acceptance Test Plan *(final)*
- ❑ Pre-acceptance Checklist
- ❑ Security Checklist
- ❑ Error Reporting and Tracking System *(optional)*
- ❑ Requirements Traceability Matrix *(final)*
- ❑ Maintenance Plan *(draft)*
- ❑ Project Plan *(revised)*

Key Activities:

Conduct Integration Testing
Conduct System Testing
Initiate Acceptance Process
Conduct Acceptance Test Team Training
Develop Maintenance Plan
Revise Project Plan
Conduct Acceptance Test
Conduct Acceptance Process
Beta Test Product
Conduct Structured Walkthroughs
Conduct In-Phase Assessment
Conduct Software Integration and Testing Phase Exit

Methods and Tools

Structured Walkthroughs

Peer Reviews

In-Phase Assessment

Quality Reviews

Phase Exit

Forms Utilized

Pre-Acceptance Checklist
Pre-Acceptance Security Issues Checklist
Software Maintenance Plan

Section 9: Integration and Testing Phase

Overview

Description:	Software integration and testing activities focus on interfaces between and among components of the software product, such as functional correctness, system stability, overall system operability, system security, and system performance requirements (e.g., reliability, maintainability, and availability). Software integration and testing performed incrementally provides feedback on quality, errors, and design weaknesses early in the integration process. In this phase, software components are integrated and tested to determine whether the software product meets predetermined functionality, performance, quality, interface, and security requirements. Once the software product is fully integrated, system testing is conducted to validate that the software product will operate in its intended environment, satisfies all customer requirements, and is supported with complete and accurate operating documentation.
Review Process:	Quality reviews are necessary during this phase to validate the product and associated deliverables. The activities that are appropriate for quality reviews are identified in this section and the Lifecycle Model section. The time and resources needed to conduct the quality reviews should be reflected in the project resources, schedule, and work breakdown structure.
SDLC References:	*Section 2 Lifecycle Model, Quality Reviews*, provides an overview of the Quality Reviews to be conducted on a project. *Section 12, Emergency Maintenance*, provides an overview of the software emergency maintenance process. *Appendix C, Conducting Structured Walkthroughs,* provides a procedure and sample forms that can be used for structured walkthroughs. *Appendix D, In-Phase Assessment Process Guide*, provides a procedure and sample report form that can be used for in-phase assessments. *Appendix E, Phase Exit Process Guide*, provides a procedure and sample report form that can be used for phase exits.

.

Conduct Integration Testing

Responsibility:

Project Team Programmers

Description:

During software integration, the software components developed by the programming staff, off-the-shelf software purchased from vendors and reusable code or modules obtained from other sources are assembled into one software product. Each assembly is tested in a systematic manner in accordance with the Integration Test Plan. An incremental approach to integration enables verification that as each new component is integrated, it continues to function as designed and both the component and the integrated product satisfy their assigned requirements.

Integration of Components:

1) *Software components;*
2) *Off-the-shelf software;*
3) *Reusable code or modules*

Integration testing is a formal procedure that must be carefully planned and coordinated with the completion dates of the unit-tested modules. Integration testing begins with a software structure where called sub-elements are simulated by stubs. A stub is a simplified program or dummy module designed to provide the response (or one of the responses) that would be provided by the real sub-element. A stub allows testing of calling program control and interface correctness. Integration testing precedes unit-tested modules or builds as replace stubs. This process continues one element at a time until the entire system has been integrated and tested.

Integration testing may be performed using "bottom up" or "top down" techniques. Most integration test plans make use of both bottom-up and top-down techniques.

Scheduling constraints and the need for parallel testing will affect the test approach.

The bottom-up approach incorporates one or more modules into a build; tests the build; and then integrates the build into the software structure. The build normally Comprises a set of modules that perform a major function of the software system. Initially, a stub that is replaced when the build is integrated may represent the function.

In the top-down approach, individual stubs are replaced so that the top-level control is tested first, followed by stub replacements that move downward in the software structure. Using top-down integration, all modules that comprise a major function are integrated, thereby allowing an operational function to be demonstrated prior to completion of the entire system.

Each requirement identified in the Software Requirements Specification must be tested during integration testing. This traceability ensures that the software product will satisfy all of the requirements and will not include inappropriate or extraneous functionality. Expand the Requirements Traceability Matrix developed in the Requirements Definition Phase to relate the integration test to the requirements. Place a copy of the expanded matrix in the Project Notebook.

Conduct Integration Testing

Description Continued:	At the completion of each level of integration testing, a test report is written. The report documents test results and lists any discrepancies that must be resolved before the tested components can be used as the foundation for another integration level. Place a copy of all integration test materials in the Project Test File.
Deliverables:	A final test report is generated at the completion of integration testing indicating any unresolved difficulties that require management attention. Place a copy of the final Integration Test Report in the Project Notebook.
Optional Deliverables:	A formal reporting system by which detected errors and discrepancies are recorded and fully described is recommended. These reports will help to confirm that all known errors are fixed before implementation of the completed software product. Error reports also help to trace multiple instances of the same error or anomalous behavior, so that error correction and prevention assignments can be implemented. The Quality Assurance representative assigned to the project can provide assistance in developing and using an error reporting/tracking system.
Review Process:	Conduct a structured walkthrough of the Requirements Traceability Matrix and final Integration Test Report.

Conduct System Testing

Responsibility:

Project Team or Independent Test Team

Description:

During system testing, the completely integrated software product is tested to validate that the product meets all requirements. System response timing, memory, performance, security, and the functional accuracy of logic and numerical calculations are verified under both normal and high-load conditions. Query and report capabilities are exercised and validated. All operating documents are verified for completeness and accuracy.

System testing is conducted on the system test bed using the methodology and test cases described in the System Test Plan. The system test bed should be as close as possible to the actual production system. Either the project team or an independent test team conducts system testing to assure that the system performs as expected and that each function executes without error. The results of each test are recorded and upon completion included as part of the project test
documentation.

When errors are discovered, the test team leader to determine the severity and necessary subsequent action should review them. If appropriate, minor problems can be corrected and regression tested by the project team programmers within the time frame allotted for the system test. Any corrections or changes to the software product must be controlled under configuration management. Major problems may be cause to suspend or terminate the system test, which should then be rescheduled to begin after all of the problems are resolved.

Customers may be encouraged to participate in the system tests to gain their confidence in the software product and to receive an early indication of any problems from the customer's perspective. Inform customers that errors and discrepancies may occur during testing and explain the error correction, configuration management, and retest processes.

Deliverables:

At the successful conclusion of system testing, the software product is ready for installation and acceptance testing. Review the draft versions of the operating documents, Training Plan, and Installation Plan. Update the documents as needed. Deliver the final versions of the operating documents, Training Plan, and Installation Plan to the system owner and customer for review and approval. Place a copy of the approved documents in the Project Notebook. Place a copy of all system test materials (e.g., inputs, outputs, results, and error logs) in the Project Test File.

Generate a test report at the conclusion of the system test process. The report documents the system test results and lists any discrepancies that must be resolved before the software product is installed and prepared for acceptance testing. Place a copy of the report in the Project Notebook.

Review Process:

Conduct a structured walkthrough of the system test report.

Initiate Acceptance Process

Responsibility:

Project Manager

Description:

The acceptance process is used to officially accept new or modified software products that satisfy the project requirements and are fully operational. The initiation of the acceptance process begins after the successful completion of system testing. Prior to the initiation of the acceptance process, review the draft Acceptance Test Plan. Make any additions or changes needed to assure that the test plan reflects the current version of the software requirements.

The acceptance process is initiated with the completion of a Pre-acceptance Checklist. This list helps to ensure that all necessary pre-acceptance activities have been completed and that the required operating documents were developed and approved. The Pre-acceptance Security Issues Checklist helps to ensure that security issues were addressed and should be completed by the system owner, as appropriate.

As part of the acceptance process, ensure that all documented issues identified during previous quality reviews have been resolved. Also, the support team should have been identified, including Hotline support. Copies of the approved operating and project documents should be provided to the support team who will maintain the system once it is accepted and transitioned to operational status. An operational analysis of the project should be conducted for support issues.

Steps:

Use the following steps to initiate the acceptance process:

- The project manager notifies the Quality Assurance representative assigned to the project that the project is ready to start the acceptance process.
- The Quality Assurance representative sends the Pre-acceptance Checklist and Pre-acceptance Security Issues Checklist to the project manager.
- The project manager completes the checklists, obtains the concurrence signature of the system owner (if required), and returns the completed checklists to the Quality Assurance representative.
- The Quality Assurance representative schedules an initial acceptance process meeting. More than one meeting may be necessary to accommodate customers at different locations or with varying requirements.

Deliverables:

Each requirement identified in the Software Requirements Specification must be tested during acceptance testing. This traceability ensures that the software product has satisfied all of the requirements and does not include inappropriate or extraneous functionality. Expand and finalize the Requirements Traceability Matrix developed in the Requirements Definition Phase to relate the acceptance test to the requirements. Place a copy of the expanded matrix in the Project Notebook.

Initiate Acceptance Process

Deliverables Continued:	Review the draft version of the Acceptance Test Plan, and update as needed. Deliver the final version of the Acceptance Test Plan to the system owner, customer, and other project stakeholders for review and approval prior to conducting any acceptance tests. Place a copy of the approved Acceptance Test Plan in the Project Notebook. After the Quality Assurance representative supporting the project reviews the Pre-acceptance Checklist and Pre-acceptance Security Issues Checklist, place copies in the Project Notebook.
Review Process:	Conduct structured walkthroughs of the Requirements Traceability Matrix, Pre-acceptance Checklist and Pre-acceptance Security Issues Checklist.
Resource:	The Pre-acceptance Checklist and Pre-acceptance Security Issues Checklist are available on the Research and Policy Web site at: http://www.michigan.gov/dit.

Conduct Acceptance Test Team Training

Responsibility:	Project Team
Description:	If the project team is not conducting the Acceptance Test, training may be required for the personnel performing the testing. The acceptance test participants and their experience with the software product and the operating environment should have been identified in the Acceptance Test Plan. The level of training will depend on the testers' familiarity with the software product and the platform on which the software will run. The advantage of having customer's acceptance test the software product is that they are the experts most familiar with the business information flow and how the software product must fit into the workplace. It is recommended that the operating documents and other test materials be distributed to the test team prior to the actual start of the acceptance test training. This will give the test team time to become familiar with the software product and the test process and procedures.
Deliverables:	Not applicable.
Review Process:	Not applicable.

Develop Maintenance Plan

Responsibility: Project Manager

Description:

The purpose of the Maintenance Plan is to determine the scope of the maintenance effort, identify the maintenance process and tools, quantify the maintenance effort (personnel and resources), and identify anticipated maintenance requirements. The Maintenance Plan needs to define the maintenance process and its boundaries or scope. The maintenance process beginning point should be defined (e.g., receipt of a change request or planned COTS version upgrade) and the ending action should be defined (e.g., implementation and sign-off of a product). The process is a natural outgrowth of many of the configuration management procedures. A description of the overall flow of work within the maintenance process should be included. The maintenance process can be tailored to the type of maintenance being performed and can be divided in several different ways. This can include different processes for corrections or enhancements, small changes or large changes, etc.

The maintenance requirements need to be identified and quantified (sized) in the Maintenance Plan to determine the future maintenance load for the organization.

The following issues should be considered when defining the requirements:

- Expected external or regulatory changes to the software
- Expected internal changes to support new requirements
- Requirements deferred from current project to later release
- Wish-list of new functions and features
- Expected upgrades for performance, adaptability, connectivity, etc.
- New lines of business that need to be supported
- New technologies that need to be incorporated

The requirements for the maintenance staff also need to be established. At this phase, the maintenance plan should address the following:

- Number of maintainers, their job descriptions, and required skill levels
- Experience level of the maintenance staff
- Documented maintenance processes at the systems and program levels
- Actual methods used by programming staff
- Tools used to support the maintenance process
- Current work load and estimates of future needs

An important part of the maintenance plan is an analysis of the hardware and software most appropriate to support the maintenance organization's needs. The development, maintenance, and test platforms should be defined and differences between the environments described. Tools sets that enhance productivity should be identified and provided. Tools should be accessible to all that need them, and sufficient training provided so that their use will be well understood.

Develop Maintenance Plan

Description continued:	Although all systems need maintenance, there comes a time when maintenance is no longer technically or fiscally viable. Issues such as resources, funds, and priorities may dictate that a system should be replaced rather than changed. The maintenance plan should identify the criteria that indicate the software product is ready for retirement or replacement, such as the failure rate, age of code, and incompatibility with current technology.
Deliverables:	Develop the Maintenance Plan. Place a copy of the draft Maintenance Plan in the Project Notebook.
Review Process:	Conduct a structured walkthrough to ensure that the Maintenance Plan accurately reflects the necessary information. The Maintenance Plan is formally reviewed during the In-Phase Assessment and Phase Exit processes.
SDLC Reference:	Refer to *Section 12, Emergency Maintenance*, for more information on maintaining the software.
Resource:	A template for the Software Maintenance Plan is available on the Research and Policy Web site at: http://www.michigan.gov/dit.

Revise Project Plan

Responsibility:	Project Manager
Description:	Once the integration and system tests are completed, determine if the project estimates for resources, cost, and schedule need to be revised.
Deliverables:	Review the Project Plan for accuracy and completeness of all Software Integration and Testing Phase activities and make any changes needed to update the information. Expand the information for the Installation and Acceptance Phase to reflect accurate estimates of resources, costs, and hours. Place a copy of the revised Project Plan in the Project Notebook. ***Note:*** A Project Plan is an effective management tool that is recommended for all projects regardless of size. The plan can be consolidated for small projects.
Review Process:	Conduct a structured walkthrough to ensure that the Project Plan reflects the project's current status and adequately estimates the resources, costs, and schedule for the Installation and Acceptance Phase. The Project Plan is formally reviewed during the In-Phase Assessment and Phase Exit processes.

Conduct Acceptance Test

Responsibility:

Acceptance Test Team

Description:

1) Obtain acceptance of software;
2) Prepare a formal acceptance test report

Acceptance of a delivered software product is the ultimate objective of a software development project. Acceptance testing is used to demonstrate the software product's compliance with the system owner's requirements and acceptance criteria.

At the system owner's discretion, the project team may perform acceptance testing, by the system owner and customers with support from the project team, or by an independent verification and validation team. Whenever possible, customers should participate in acceptance testing to assure that the software product meets the customers' needs and expectations. All acceptance test activities should be coordinated with the system owner, customer(s), operations staff, and other affected organizations.

Acceptance testing is conducted using acceptance test data and test procedures established in the Acceptance Test Plan. Testing is designed to determine whether the software product meets functional, performance, and operational requirements. If acceptance testing is conducted on an incremental release basis, the testing for each release should focus on the capabilities of the new release while verifying the correct operation of the requirements incorporated in the previous release.

Acceptance testing usually covers the same requirements as the system test.

Acceptance testing may cover additional requirements that are unique to the operational environment. The results of each test should be recorded and included as part of the project test documentation.

Subject the test environment to strict, formal configuration control to maintain the stability of the test environment and to assure the validity of all tests. Review the acceptance test environment, including the test procedures and their sequence, with the system owner and customer before starting any tests.

Testing is complete when all tests have been executed correctly. If one or more tests fail, problems are documented, corrected, and retested. If the failure is significant, the acceptance test process may be halted until the problem is corrected.

Deliverables:

Prepare a formal Acceptance Test Report. Summarize the test procedures executed, any problems detected and corrected, and the projected schedule for correcting any open problem reports. Place a copy of all acceptance test materials in the Project Test File.

Review Process:

Conduct an Operational Readiness Review at the completion of acceptance testing. This review is a combined quality assurance and configuration management activity. It focuses on the results of the acceptance test and the readiness of the software product to go into production.

Conduct Acceptance Test

Review Process Continued:

The Operational Readiness Review includes a functional configuration audit to determine whether the test records demonstrate that the product meets its technical requirements, and a physical configuration audit to determine whether the product technical documentation is complete and accurately describes the software product.

During the Operational Readiness Review examine acceptance test results with the system owner and customer. Document any problems, determine solutions to the problems, and implement action plans. Once any problems associated with the acceptance test are resolved, the software product is ready for formal acceptance by the system owner.

A successful Operational Readiness Review establishes the operational baseline for the software product. The operational baseline is the final baseline. It consists of the software product and the technical documentation that describes the operational software and its characteristics. It contains the current functional baseline, the product baselines for the configuration items comprising the system, and other system-level technical documentation generated during the lifecycle.

If the operational product requires enhancements or changes to correct problems, each new release should be preceded by an Operational Readiness Review, after which the updated system documentation is established as a new operational baseline superseding the previous one.

Conduct Acceptance Process

Responsibility:

Project Manager

Description:

The acceptance process is used to officially accept new or modified software products that satisfy the customers' requirements and are fully operational. The acceptance process is concluded when the acceptance test has been successfully completed, the software product has been installed and is operational at all customer sites, and complete operating documentation describing the product has been approved and delivered.

At the conclusion of the acceptance process, responsibility for the software product is formally transferred from the project team to the system owner and maintenance staff.

Steps:

Use the following steps to conclude the acceptance process:

- The project manager notifies the Quality Assurance representative assigned to the project that the software product is ready to complete the acceptance process.
- The Quality Assurance representative sends the Acceptance Checklist to the project manager.
- The project manager completes the checklist, obtains the concurrence signature of the system owner (if required), and returns the completed checklist to the Quality Assurance representative.
- The Quality Assurance representative schedules an acceptance meeting. More than one meeting may be necessary to accommodate customers at different locations or with varying requirements.

Deliverables:

The Acceptance Checklist is completed and submitted to the Quality Assurance representative supporting the project. This list helps to ensure that all necessary acceptance activities have been completed and that the required operating documents were developed and approved.

The system owner to verify that the software product is acceptable and ready for production develops a formal written acceptance of the software product.

Review Process:

Conduct structured walkthroughs or peer reviews of the Acceptance Checklist.

Resource:

An Acceptance Checklist is available on the Research and Policy Web site at: http://www.michigan.gov/dit.

Alpha, Beta, Gamma Test Product

Responsibility:

Description:

The purpose of the integration and testing phase is to *build the application* and *and test it* to verify that it works as stated in the design specifications and prototype. Typically the application is developed through a series of fast builds, which are often called *alpha, beta and gamma,* especially in multimedia production:

1. **Alpha:** the first full implementation of the original application design
2. **Beta**: modifications and changes based on usability testing feedback, whereupon the application design is frozen
3. **Gamma**: final programming and data preparation

The building and testing is an *iterative process* of coding, testing, correcting defects, and re-testing.

Step 1: Alpha:

"Alpha" is used to refer to the segment during which the first complete implementation of application takes place, and which results in the "alpha version" of the application. The alpha version represents the first overall completion of the program functionality, i.e. it contains (nearly) full functionality but may not contain all the final content. The alpha version is often not rigorously tested for software defects.

Version Implementation:

In larger projects, the alpha step is divided into several mini milestones called *versions*. Each new version contains additional modules from the application design. In the example below, delivery of V.04 would correspond to delivery of the "alpha version" and thus would complete the alpha segment.

	Main page	Feature Set 1	Feature Set 2	Feature Set 3	Miscellaneous Features
Version 01	x	x			
Version 02	x	x	x		
Version 03	x	x	x	x	
Version 04	x	x	x	x	x

Alpha, Beta, Gamma Test Product

Version Implementation Continued:

The version implementation approach effectively breaks the application into discrete modules. This approach has several advantages, including:

- **Independent implementation**: several developers can *work on the program simultaneously* and thereby helps to minimize the overall development time.
- **Error containment**: individual developers work on individual modules, errors can be contained to those particular modules (rather than spreading the same errors throughout the application).
- **Unit testing**: completing each module independently permits testing to be done much earlier in the process.

Keeping bug lists:

By testing the individual modules early in the development phase, you will get immediate feedback on the quality of the code. The feedback can be of great use in keeping track of errors as they are found and fixed. The best way to keep track of software defects (bugs) is with a database in which each defect is entered, along with information that pertains to it, such as:

- a unique identification number
- a description the defect
- where it occurred
- which version of the program it was found in
- relative severity
- instructions on how to replicate it
- the date is was reported
- current status
- who reported it
- and so on

When categorizing the bugs, it is helpful to make a distinction between software errors, content errors, and design issues.

Starting Documentation:

The alpha segment is the right time to start writing the user guide and other system documentation (e.g. System Operation Guide, Technical Reference). The guide should be outlined based on the planning done in the application design phase. It can even be written, to some extent, while allowing for the fact that it will have to revised if there are design changes.

It's good idea to start on the guide early, because if you wait until the beta segment, time will be short, and the software may be completed first.

Testing:

The alpha version represents the earliest opportunity to show the customer and users the actual application up and running. The sooner you can find any problems in the design, the easier they can be fixed. Software defects are to be expected in the alpha version.

The purpose of such alpha users testing is primarily to *make sure the application will meet its requirements, not to test the reliability of the software*.

Alpha, Beta, Gamma Test Product

Testing Continued:

Because of the bugs of the alpha version, the user testing should be conducted (or at least started) by having a demonstrator or facilitator to operate the application, rather than asking the users to do it on their own. The alpha segment is a good time to start writing quality assurance testing specifications from the application design. The goal is to create a detailed checklist of all functions and features contained in the application, to be used in the full quality assurance testing (which may begin as early as the beta step).

Significance of the Alpha Version:

The alpha version is of significant *psychological benefit*, because it is the first time team members actually see the multimedia application up and running. The application is suddenly "real", after the long and sometimes painful period of specifications and design.

Another significant benefit of the alpha version is that you will learn *if the chosen hardware and software platform can support the user interface and application features as designed*.

During the alpha segment, team members working with data preparation will produce samples to be used in the alpha version. By doing this, the team will be able *to spot any potential problems and concerns related to the creation and conversion of the required data elements*.

Accepting the Alpha Version:

Alpha segment ends when the alpha version is completed and accepted by the client. This sounds simple, but usually it is quite difficult to agree when the application has reached the alpha segment. This is because 1) many design changes compared with the original requirements specification, and 2) the difficulty of implementing all of the functionality specified when the specification is continuously changing as the development work progresses. Finally, the decision if a version qualifies as alpha, will be made by the client.

Step 2 Beta:

Once the alpha version has been accepted, it must be evaluated. This usually leads to a *period of re-design and modification*. The period of modification and final design is referred to as the beta segment of the development phase. The beta segment ends with the acceptance of the final beta version, in which the design changes and modifications have been implemented. At this point *the design is "frozen"*, with little opportunity for anyone to modify it.

Freeze the Features:

During the beta segment, any remaining design modifications should be resolved as soon as possible. To avoid time consuming and costly changes in the final application, the modifications should be proposed and decided upon early in the beta segment, and they should be kept to an minimum. The beta segment is one of the riskiest parts of a project.

In the end, however, a time comes during the development of any application when the design must be frozen; otherwise it will never be finished. Instead of accepting new modifications, a good approach is to start to use a ***"wish list"*** for the next version of the application.

Alpha, Beta, Gamma Test Product

Final Customer Interface:

Beta segment is the last (and worst) chance to make modifications to the customer interface. Any changes made after the final beta version is complete will usually have serious schedule and cost consequences, because late changes are disproportionately expensive and time-consuming. Therefore:

1. support only the most important and easiest-to-implement changes
2. if the changes are really important, make sure they get added

Minimizing Features Creep:

While it is very difficult to avoid the feature creep completely, it can be minimized in at least three ways:

1. define the cost to implement the desired feature in the quantifiable, objective terms of time and money
2. give an individual suggesting a new feature the option of putting it on a wish list for next version
3. "just say no" if appropriate and possible.

Procedure Manual:

During the beta segment, the design and writing of procedure manual and the packaging artwork should be well underway. While the procedure manual can be started early based on the application design, if design changes are accepted during beta, the user guide will have to be altered to reflect those changes.

For information on what to include in the user guide, please refer to the *Guidelines for Procedure Manual* at end of phase.

Technical Documentation:

Any design flaws uncovered or changes made should be well documented, and the application design documents updated to include those changes. Simply implementing the changes in the application without adding them to the documents may create significant problems in the current project and will certainly cause problems in maintaining and updating the product in the future.

For information on what to include in the technical documentation, please refer to the *Guidelines for Technical Reference* at end of phase.

Step 3: Gamma:

Once beta version has been completed, *data preparation* must be finished and the application subjected to extensive *testing*. This part of the development phase is often called the gamma segment. Besides data preparation, tasks such as packaging, user guide and technical documentation need to be finished up as well.

The gamma phase is marked by *extensive attention to detail* and keeping track of software defects and content errors.

Completing Data Preparation:

Finishing data preparation requires checking all the various data elements, including the content of any text, video, audio, graphics, illustrations, animation and numerical data. This includes e.g. proofreading all the texts and fixing any typos; linking photos into articles, attaching captions, cleaning up graphics and so on.

Alpha, Beta, Gamma Test Product

Testing:

Although the testing should be done all the way during the development phase, the testing and debugging often forms a tedious and time consuming part of the gamma segment.

The testing process can be divided into four major segments (with unit testing being carried out in alpha and beta segments as well):

1. unit testing - testing of smallest logical *units* of the application
2. integration testing - verifying that multiple *modules* of application work together
3. system testing - verifying that the *complete application* (or systems) works
4. user acceptance testing - users validate that all elements of the application (hardware, software, user interface, documentation) meets their requirements

Documentation:

By the time gamma starts, the user guide and the technical documentation should be well underway; by the end of the gamma phase, they should be stabilized and screen shots can be gathered and incorporated into them. For information on what to include in the documents, please refer to:

- Guideline for Procedure Manual
- Technical Reference Guide

Conduct Project Reviews

Conduct Structured Walkthroughs

Responsibility:	Project Manager, Work Product Author and Reviewers
Description:	During the Software Integration and Testing Phase, schedule at least one structured walkthrough to review each of the Software Integration and Testing Phase deliverables, i.e., Integration Test Reports, System Test Report, Operating Documents, Training Plan, Installation Plan, Acceptance Test Plan and Pre-acceptance Checklist. ***Note:*** A structured walkthrough is an effective project management tool that is recommended for all projects regardless of size; however, if an item has gone through a prior formal structured walkthrough, and the modifications are minor, a less formal review may be warranted.
SDLC Reference:	*Appendix C, Conducting Structured Walkthroughs*, provides a procedure and sample forms that can be used for structured walkthroughs.

Conduct In-Phase Assessment

Responsibility:	Project Manager and Independent Reviewer
Description:	Schedule at least one IPA prior to the Software Integration and Testing Phase Exit process. Additional IPAs can be performed during the phase, as needed. Periodic reviews of the integration and system test results and logs are recommended. ***Note:*** An IPA is an effective project management tool that is recommended for all projects regardless of size.
SDLC Reference:	*Appendix D, In-Phase Assessment Process Guide,* provides a procedure and sample report form that can be used for In-Phase Assessments.

Conduct Software Integration and Testing Phase Exit

Responsibility:	Project Manager
Description:	During the Phase Exit meeting, participants discuss open issues that will impact the Project Plan. The project manager should ensure that an acceptable action plan is developed for handling all open issues. At the conclusion of the meeting, concurrence is needed from the designated approvers to begin the next phase. ***Note:*** A Phase Exit is an effective project management tool that is recommended for all software projects regardless of size. For small software projects, phases can be combined and addressed during one Phase Exit.
SDLC Reference:	*Appendix E, Phase Exit Process Guide,* provides a procedure and sample report form that can be used for phase exits.

Guidelines for Procedure Manual

This document provides guidelines for issues that should be covered in the Procedure Manual. The Procedure Manual should provide description of all critical features and functions of the application and how to use them.

NOTE: You have to pick up issues relevant to your project. The bigger the application to be produced, the more detail you have to cover in the User Guide document.

Cover page

Table of Contents

Introduction

- Purpose of the guide
- Intended audience
- How to use the guide
- General conventions used in the guide

Application Overview

- Application objectives
- Situation of the system within the organization
    - System context diagram
- Description of major functional components

Graphic User Interface

- Introduction
- General menu/toolbar options
- Online help
- Descriptions of main screens
- Data field and control descriptions
- General task flow diagram
- Task descriptions
    - why, when, how, error handling

Appendix

Guidelines for Technical Reference Guide

This page provides guidelines for issues that should be covered in the Technical Reference Guide. The Technical Reference Guide should document the core technical components of the system for those who will maintain and/or develop the system.

***NOTE**: You* have to pick up issues relevant to your project. The bigger the application to be produced, the more detail you have to cover in the Technical Reference Guide.

Cover page

Table of Contents

Introduction

- Purpose of the guide
- Intended audience
- How to use the guide
- General conventions used in the guide

System Overview

- Technical architecture and requirements
    - Architecture description and diagram
    - Hardware platform
    - Software platform
    - Database server
- General description of major system components

System Components

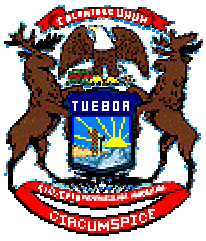
- Web site structure
    - Logical information structure
    - Physical directory structure
- Database
    - Structure (ER-diagram)
    - SQL constructs for tables, indexes, views, ...
- Software modules
    - Description
    - Diagram
    - Source code

Installation

- Source location(s)
- Installation procedure

Known errors and problems

Ideas for further development



SYSTEMS DEVELOPMENT LIFECYCLE

SECTION 10

INSTALLATION AND ACCEPTANCE PHASE

Section 10: Installation and Acceptance Phase

Table of Contents

Section 10: Installation and Acceptance Phase

Highlights of Phase

Installation and Acceptance Phase Highlights

Inputs:

- Integration Test Materials
- System Test Materials
- Operating Documents
  - Procedure Manual
  - Programmers Reference Manual
- Training Plan
- Installation Plan
- Conversion Plan
- Acceptance Test Plan
- Pre-acceptance Checklist
- Security Checklist
- Maintenance Plan *(draft)*
- Project Plan *(revised)*
- Software Quality Assurance Plan
- Transition Plan

Outputs:

- Converted data or system files
- Installation Test materials
- Customer Training materials
- Operational software product
- Operating documents
- Maintenance Plan *(final)*
- Project Plan *(revised)*

Key Activities:

Perform Installation Activities
Conduct Installation Tests
Conduct Customer Training
Transition to Operational Status
Revise Maintenance Plan
Revise Project Plan
Conduct Structured Walkthrough(s)
Conduct In-Phase Assessment
Conduct Installation and Acceptance Phase Exit

Methods and Tools

Complete documentation for:

- computer operators
- data entry personnel/end users
- maintenance programmers
- management

Install system:

- one of four approaches may be used
- log all errors/problems to aid in maintenance

Section 10: Installation and Acceptance Phase

Overview

Description:

1) *Installation and acceptance of software product;*
2) *Install software and databases onto hardware;*
3) *Verify that software meets design;*
4) *Obtain acceptance and approval;*
5) *Customer training*

Installation and acceptance of the software product are initiated after the system test has been successfully completed. This phase involves the activities required to install the software, databases, or data that comprise the software product onto the hardware platform at the site(s) of operation. The objectives of the activities in this phase are to verify that the software product meets design requirements and to obtain the system owner's acceptance and approval of the software product. The activities associated with this phase should be performed each time the software product is installed at an acceptance test site or production site.

Customer training may be required to complete the installation process. A description of the training necessary for programmers, testers, customers, and operations staff is provided in the Training Plan.

Review Process:

Quality reviews are necessary during this phase to validate the product and associated deliverables. The activities that are appropriate for quality reviews are identified in this section and the *Section 2 Lifecycle Model*. The time and resources needed to conduct the quality reviews should be reflected in the project resources, schedule, and work breakdown structure.

SDLC References:

Section 2 Lifecycle Model, Quality Reviews, provides an overview of the Quality Reviews to be conducted on a project.

Appendix C, Conducting Structured Walkthroughs, provides a procedure and sample forms that can be used for structured walkthroughs.

Appendix D, In-Phase Assessment Process Guide, provides a procedure and sample report form that can be used for in-phase assessments.

Appendix E, Phase Exit Process Guide, provides a procedure and sample report form that can be used for phase exits.

Perform Installation Activities

Responsibility: Project Team

Description:

The installation process involves loading, copying, or migrating the software and data, if required, to the production platform and the provision of operating documentation and other support materials at each site. The installation of firmware, hardware, and communications equipment may also be involved.

1) Migrate software and data;
2) Install firmware, hardware, and communications equipment;
3) Implement system and data conversion;
4) Compare new software with current system;
5) Site preparation completed;
6) *Conduct inventory*

If a current system exists, implement system and data conversion in accordance with the procedures described in the Conversion Plan. Each data and file conversion should include a confirmation of data and file integrity. Determine what the output in the new software product should be compared with the current system, and assure that the data and files are synchronized.

At each installation site, inspect the facility to assure that site preparation is complete and in accordance with the Installation Plan. Initiate any actions that are needed to complete the preparations. Conduct an inventory of all vendor provided hardware, software, firmware, and communications equipment in accordance with the Acquisition Plan.

Follow the steps specified in the Installation Plan when installing the software, hardware, and other equipment. Monitor all installation activities including those performed by vendors.

Steps:

Use the following steps to perform the installation activities:

- ❑ Coordinate the installation with the system owner, customers, operations staff, and other affected organizations.
- ❑ Ensure that any necessary modifications to the physical installation environment are completed.
- ❑ Inventory and test the hardware that will support the software product. This inventory should be performed in advance of the planned installation date to allow time for missing hardware to be obtained and malfunctioning equipment to be replaced or repaired.
- ❑ If the software product requires an initial data load or data conversion, install and execute the tested programs to perform this process. Install the software product onto the hardware platform.

Deliverables: Not applicable.

Review Process: Not applicable.

Conduct Installation Tests

Responsibility:

Project Team

Description:

Ensure the integrity and quality of the installed software product by executing the installation tests defined in the Installation Plan. Testing is performed to verify that the software product has been properly installed and is fully operational.

1) Execute installation tests;
2) Verify software is properly installed and operational;
3) Document any problems;
4) Identify corrective action;
5) Select diagnostic package;
6) Retest all equipment and software

The installation test(s) are designed to validate all functions of the software product and should specify a standard set of test results and tolerances. If the software product being installed is a modification to an existing system, all remaining functions and code that may be affected by the new software should be tested.

Document any problems and identify corrective action. Select a diagnostic package that will pinpoint problems quickly and allow for timely corrections. Retest all equipment and software after a repair, replacement, or modification.

Certify each software component on successful completion of installation and checkout. When installation is complete, rerun a portion or all of the system test and dry run the acceptance test procedures to verify correct operation of the software product.

Conduct installation testing to verify the following:

- ❑ Security functions
- ❑ Integration with the current software
- ❑ Interfaces with other systems
- ❑ System functionality based on the requirements

Deliverables:

Place a copy of all Installation Test materials in the Project Notebook.

Review Process:

Conduct structured walkthroughs of the Installation Test materials.

Conduct Customer Training

Responsibility:	Project Team
Description:	Customer training is an important factor in the success of the operational software product. During training, most customers will receive their first hands-on experience with the software product. Operations and maintenance staff may also be trained to use, monitor, and maintain the software product. The objective of the training is to provide the trainee with the basic skills needed to effectively use the software product and to raise the customer's confidence and satisfaction with the product. The type of training will depend on the complexity of the software product, and the number and location of the customers to be trained. Alternative training formats include formal classroom training, one-on-one training, computer-based instruction, and sophisticated help screens and online documentation. Conduct the training as described in the Training Plan. Conduct a pilot test of the training session(s). Include members of the project team, the system owner, and key customers. Have all participants evaluate the training content, instruction, and support materials. Make any necessary changes to the training program prior to general customer training sessions. If consecutive training classes are conducted, feedback should be requested from the participants and used to continuously improve the training approach, methods, and materials. At the completion of the training, customers should have a thorough understanding of the input requirements of each transaction, the processing that takes place, and the types of output that are generated.
Deliverables:	Submit a copy of the training materials to the system owner and customer for review and approval. Place a copy of the approved training materials in the Project Notebook. Training materials are subject to the same configuration control procedures as the other operating documents and should remain current with changes to the software product.
Review Process:	Conduct structured walkthroughs of the training materials. The pilot test of the training session(s) helps ensures implementation of a quality-training program.

Transition to Operation Status

Responsibility:	Transition Team
Description:	The transition of the software product to full operational status begins after the formal acceptance by the system owner. Use the procedures described in the Transition Plan to implement the transition processes. Conduct or support stress tests and other operational tests. Determine product tolerances to adverse conditions, failure modes, recovery methods, and specification margins. Complete any training and certification activities. Ensure that support to be provided by contractors begins as planned. The project team is usually expected to provide operational and technical support during the transition. Identify project team personnel with a comprehensive Understanding of the software product that can provide assistance in the areas of software installation and maintenance, test, and documentation of changes. Technical support may involve the analysis of problems in software components and operational procedures, the analysis of potential enhancements, and vendor-supplied upgrades to software components (such as the operating system or data base management system). Transition to full operational status should be an event-oriented process that is not complete until all transition activities have been successfully performed. Withdraw the support of the project team personnel in a gradual sequence to ensure the smooth operation of, and customer confidence in, the software product. At the conclusion of the transition process, plan a formal transfer of all responsibility to the maintenance staff. All Project Notebook materials, operating documents, a list of any planned enhancements, and other pertinent records should be turned over to the maintenance staff. Access rules must be modified to provide access to the product by the maintenance staff and to remove access by the project team and other temporary customer accesses. Programs, files, and other support software should be in the production library and deleted from the test library, where appropriate. For major software systems involving multiple organizations and interfaces with other systems, a formal announcement of the transition to production is recommended. The announcement should be distributed to all affected groups. The names and telephone numbers of the maintenance staff should be included.
Deliverables:	The system is transitioned into operational status. Project Notebook materials, operating documents, and other pertinent records are turned over to the maintenance staff.
Review Process:	All reviews related to the functionality were completed prior to the system being placed into operational status.

Revise Maintenance Plan

Responsibility:	Project Manager
Description:	Once system installation and acceptance are complete, determine if the Maintenance Plan needs to be revised.
Deliverables:	Review the Maintenance Plan for accuracy and completeness and make any changes needed to update the information. Place a copy of the final Maintenance Plan in the Project Notebook.
Review Process:	Conduct a structured walkthrough to ensure that the Maintenance Plan reflects the necessary information. The Maintenance Plan is formally reviewed during the In-Phase Assessment and Phase Exit processes.

Records Retention Schedule

Responsibility:	Project Manager
Description:	Once system installation and acceptance are complete, determine if the project estimates for resources, cost, and schedule need to be revised.
Deliverables:	Review the Project Plan for accuracy and completeness of all Installation and Acceptance Phase activities and make any changes needed to finalize the information. Expand the information for the Maintenance Phase to reflect accurate estimates of resources, costs, and hours. Place a copy of the final Project Plan in the Project Notebook. ***Note:*** A Project Plan is an effective management tool that is recommended for all projects regardless of size. The plan can be consolidated for small projects.
Review Process:	Conduct a structured walkthrough to ensure that the Project Plan reflects the project's current status and adequately estimates the resources, costs, and schedule for the Maintenance Phase. The Project Plan is formally reviewed during the In-Phase Assessment and Phase Exit processes.

Conduct Structured Walkthroughs

Responsibility:

Project Manager, Work Product Author and Reviewers

Description:

This section describes the structured walkthroughs that are used with the systems development lifecycle. The structured walkthrough is an organized procedure for reviewing and discussing the technical aspects of software or systems development lifecycle deliverables including documentation. Structured walkthroughs are used to find errors early in the development process and to improve the quality of the product. They are very successful in identifying design flaws, errors in analysis or requirements definition, and validating the accuracy and completeness of the deliverables.

Structured walkthroughs are conducted during all phases of the project lifecycle. They are used during the development of work products that contain deliverables. Structured walkthroughs are used after the deliverables have been completed to verify the correctness and the quality of the finished product. They should be scheduled in the work breakdown structure developed for the project plan and can be referred to as code reviews, design reviews, or inspections. Structured walkthroughs should also be scheduled to review small, meaningful pieces of work. The progress made in each lifecycle phase should determine the frequency of the walkthroughs; however, they may be conducted multiple times on a deliverable to ensure that it is free of defects.

Structured walkthroughs can be conducted at various times in the development process, in various formats, with various levels of formality, and with different types of participants. They typically require some advance planning activities, a formal procedure for collecting comments, specific roles and responsibilities for participants, and have prescribed follow-up action and reporting procedures. Frequently reviewers include people outside of the developer's immediate peer group.

During the Installation and Acceptance Phase, schedule at least one structured walkthrough to review each of the Installation and Acceptance Phase deliverables, i.e., Customer Training Materials, Acceptance Test Report, and Acceptance Checklist.

Note: A structured walkthrough is an effective project management tool that is recommended for all projects regardless of size; however, if an item has gone through a prior formal structured walkthrough, and the modifications are minor, a less formal review may be warranted.

Deliverables:

A Structured Walkthrough Meeting Record is available to assist the reviewers with recording errors found prior to the walkthrough session, and for the scribe to record information discussed during the walkthrough. The Project Manager ensures the presenter or author of the deliverable completes a Structured Walkthrough Management Summary Report and a copy is placed in the Project Notebook.

SDLC References:

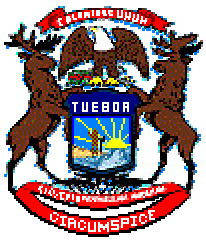
Appendix C, Conducting Structured Walkthroughs, provides a procedure and sample forms that can be used for structured walkthroughs.

Conduct In-Phase Assessment

Responsibility:	Project Manager and Independent Reviewer
Description:	An In-Phase Assessment (IPA) is an independent review of the deliverables produced or revised during each phase of the project lifecycle. The independent reviewer is typically a Quality Assurance representative who is assigned to the software project and conducts all of the IPAs for the project. An IPA does not require meetings with, or extra work by, the project team. All of the deliverables needed for the review should be readily available in the Project Notebook. Schedule at least one IPA prior to the Installation and Acceptance Phase Exit process. Additional IPAs can be performed during the phase, as needed. Provide the reviewer with copies of all deliverables developed or revised during the Installation and Acceptance Phase including the Project Plan. The reviewer assesses the deliverables to verify the following: ❑ The project is complying with the systems development standards and/or best practices. ❑ Sound project management practices are being used. ❑ Project risks are identified and mitigated. ***Note:*** An IPA is an effective project management tool that is recommended for all projects regardless of size.
Deliverables:	An IPA report form is prepared by the independent reviewer and is used to identify open issues that need to be resolved in this phase. The report is delivered to the project manager and a copy should be placed in the Project Notebook.
SDLC Reference:	*Appendix D, In-Phase Assessment Process Guide*, provides a procedure and sample report form that can be used for In-Phase Assessments.

Conduct Installation and Acceptance Phase Exit

Responsibility:	Project Manager
Description:	The Phase Exit is a process for ensuring that projects meet the Agency and project standards identified in the Project Plan. The goal of a Phase Exit is to secure the approval of designated key individuals to continue with the project and to move forward into the next lifecycle phase. Schedule the Phase Exit as the last activity of the Installation and Acceptance Phase. It is the responsibility of the project manager to notify the appropriate participants when a project is ready for the Phase Exit process and to schedule the Phase Exit meeting. All functional areas and the Quality Assurance representative involved with the project should receive copies of the deliverables developed in this phase. During the Phase Exit meeting, participants discuss open issues that will impact the Project Plan. The project manager should ensure that an acceptable action plan is developed for handling all open issues. At the conclusion of the meeting, concurrence is needed from the designated approvers to begin the next phase.
Deliverables:	A summary of the Phase Exit meeting is prepared by the project manager or a designee and distributed to the meeting attendees. The summary identifies any issues and action items needed to obtain concurrence prior to proceeding to the Maintenance Phase.
SDLC Reference:	*Appendix E, Phase Exit Process Guide*, provides a procedure and sample report form that can be used for phase exits.



SYSTEMS DEVELOPMENT LIFECYCLE

SECTION 11

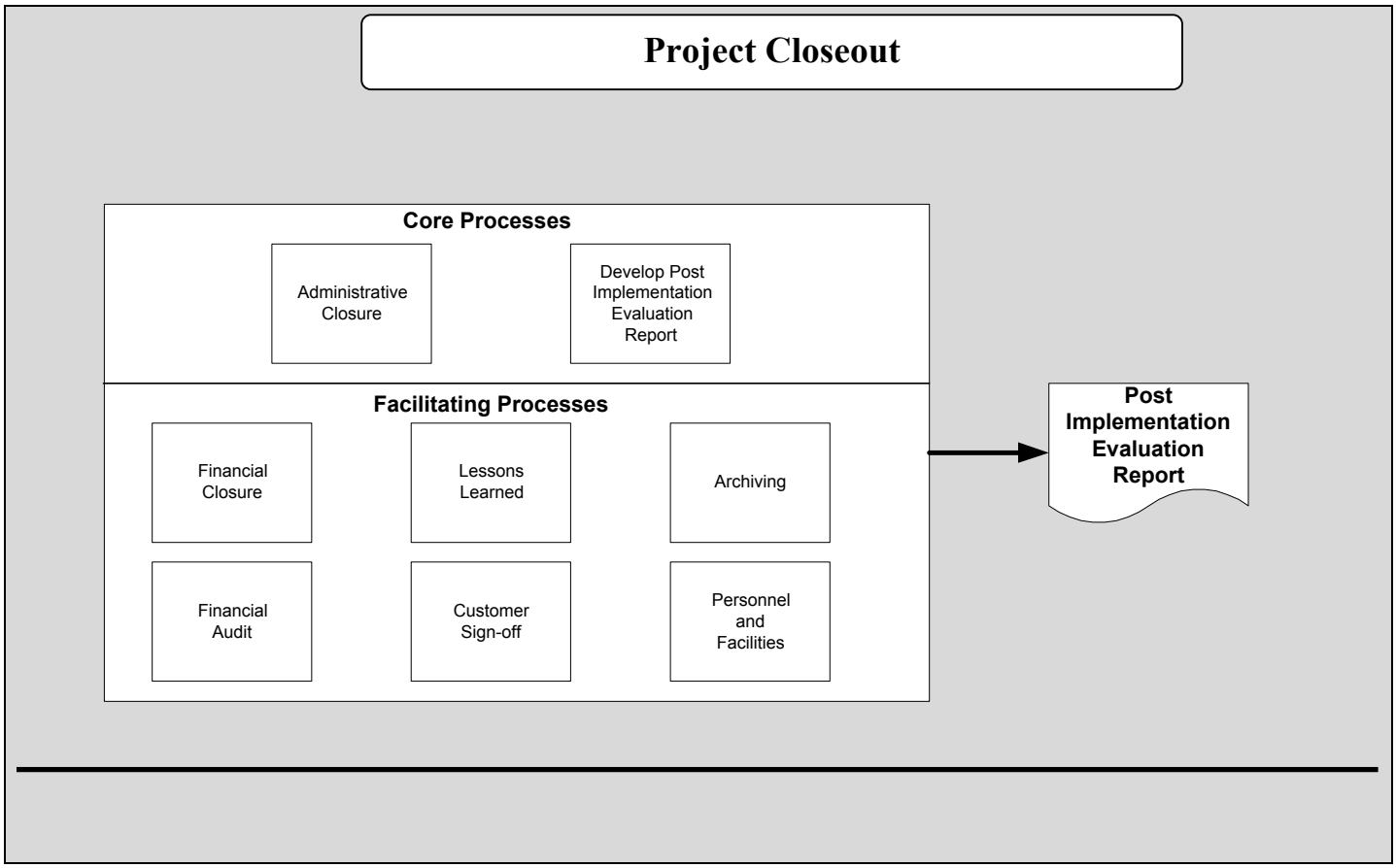
PROJECT CLOSEOUT

Section 11: Project Closeout Phase

Highlights of Phase

Section 11: Project Closeout Phase

Highlights of Phase



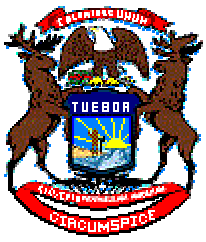
Section 11: Project Closeout Phase

Overview

Description:

This section describes the closeout activities associated with the systems development lifecycle. The closeout phase is the last phase of the lifecycle. The closeout phase is performed once all defined project objectives have been met and the customer has accepted the project's product.

The project closeout template can be found on the project management methodology Web site at: http://www.michigan.gov/dit.



SYSTEMS DEVELOPMENT LIFECYCLE

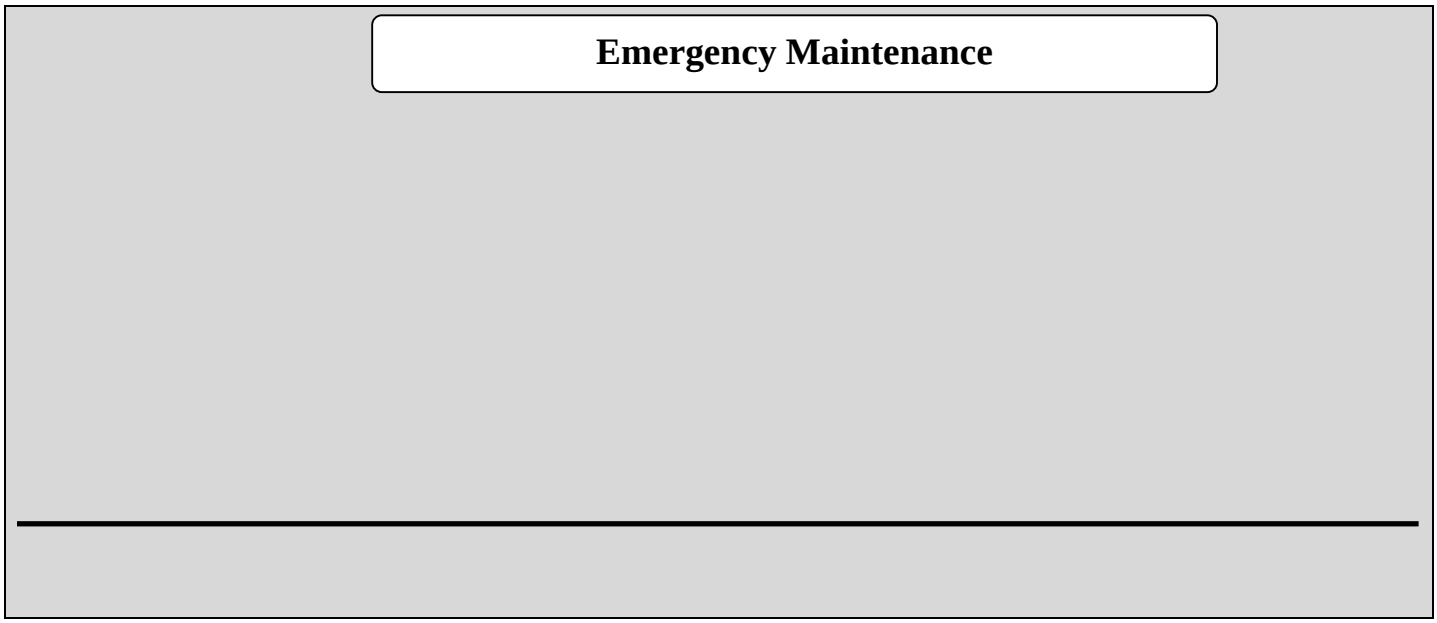
SECTION 12

EMERGENCY MAINTENANCE

Section 12: Emergency Maintenance

Table of Contents

Highlights of Phase



Section 12: Emergency Maintenance

Overview

Description:

This section describes the emergency maintenance activities associated with the systems development lifecycle. The emergency maintenance phase is performed once all defined project objectives have been met and the customer has accepted the project's product.

The process of correcting flaws and enhancing the capability of an information system.

Scope : An operational information system.

Input : Known flaws in or a request to enhance an operational information system.

Tasks :

Evaluate System

Assess Changes or Enhancement Requests

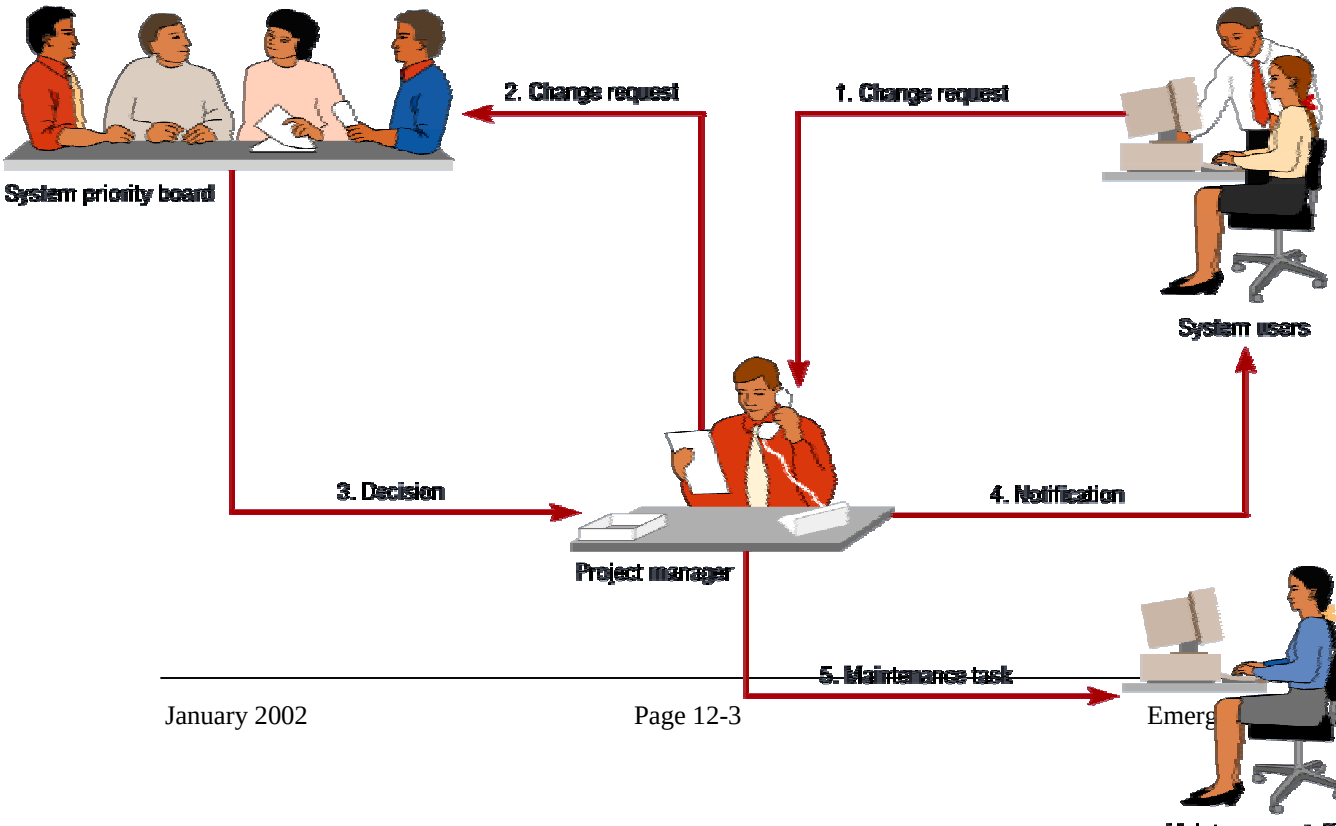
Analyze the Nature of the Change

Analyze the Impact of the Change

Execute the Change

Output : Enhanced or maintained operational information system.

HOW A MAINTENANCE REQUEST MOVES THROUGH AN ORGANIZATION



Overview

Emergency Maitnenance:

Changes made to a system to fix or enhance its functionality

Corrective Maintenance:

Changes made to a system to repair flaws in its design, coding or implementation:

Process:

- Users report bug
- IS staff evaluates, assigns
- Benchmark, test original
- Make changes
- Benchmark, test changed system
- Coordinate version ID
- Update documentation
- Report outcome to user

System Enhancement:

Types:

- *Adaptive Maintenance* - Changes made to a system to evolve its functionality to changing business needs or technologies.
- *Perfective Maintenance* - Changes made to a system to add new features or to improve performance

Process:

- Initiated by
  - User request
  - IS-driven (new technology)
  - Reengineering
- Analyze scope of change
  - Analysis
  - Design
  - Program change

Preventive Maintenance:

Changes made to a system to avoid possible future problems.

What is Software Maintenance?

Description:

Maintenance is concerned with the process used in performing software maintenance. Such a process would include phases similar to those in a process for developing a new software product. A maintenance process starts with a change request and a preliminary problem analysis. Next, a managerial and technical analysis is undertaken to investigate and determine the cost of alternative solutions. Then, the chosen solution is implemented and tested. Finally, the change is released to the customer.

Software Maintenance is:

- Changes that have to be made to computer programs after they have been delivered to the customer.
- The performance of those activities required to keep a software system operational and responsive after it is accepted and placed into production.
- Maintenance covers the life of a software system from the time it is installed until it is phased out.
- Modifications of a software product after delivery to correct faults, improve performance or other attributes, or to adapt the product to a modified environment.
- Software product undergoes modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify existing software product while preserving its integrity.

Types of Maintenance:

The common theme of the above definitions is that maintenance is an "after-the-fact" activity. Maintenance occurs after the product is in operation (during post delivery phase).

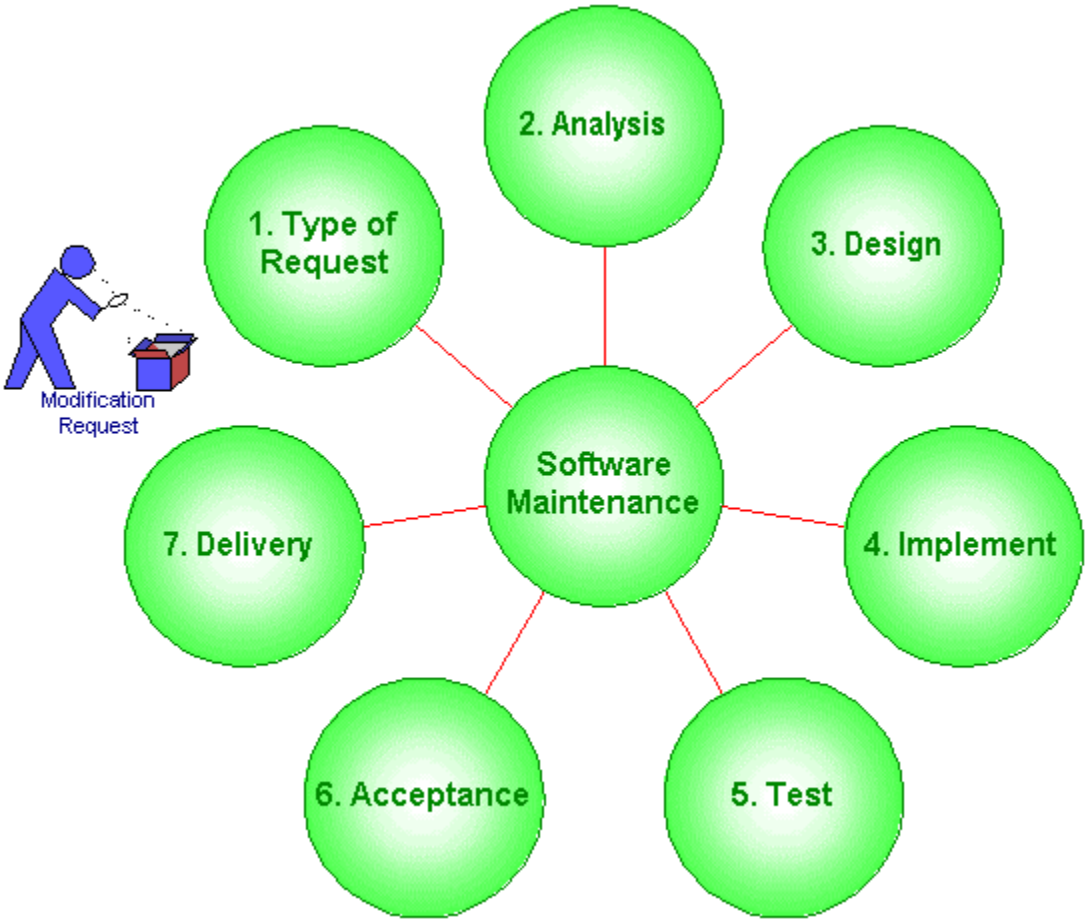
Corrective - Change to a software product after delivery to correct defects.

Adaptive - Change to a software product after delivery to keep it functioning properly in a changed or changing environment.

Emergency - Unscheduled corrective maintenance required to keep a system operational.

Scheduled - Change to a software product after delivery on a routine basis to improve performance or maintainability

The Maintenance Process



Description:

The following describes the maintenance model:

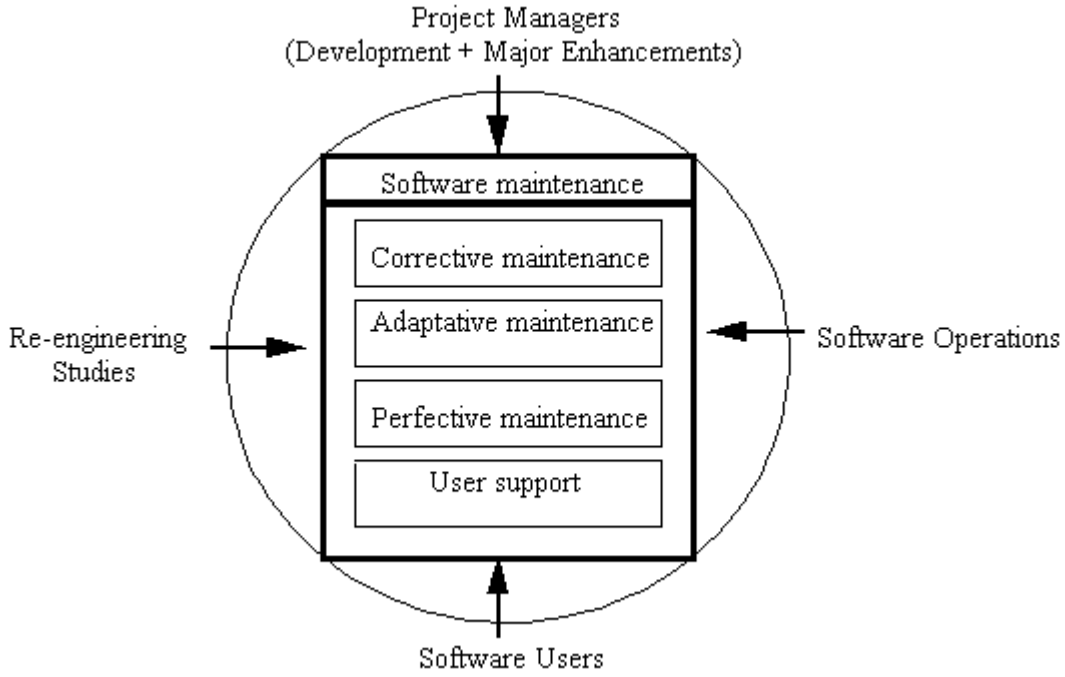
1. **Type of Request** – a request for change to the software starts the maintenance process. The request for change is submitted in the form of a modification request. The maintenance request is typically a correction to the system. Enhancements are handled by tailoring the full Systems Development Lifecycle (SDLC) to the modification request.
2. **Analysis –** during the analysis phase, a feasibility analysis is conducted. The feasibility analysis looks at items such as the impact of the modification, alternative solutions, and costs.
3. **Design –** during the design phase, all of the information that has been gathered up to this point is now reviewed and is used to design the modification.
4. **Implementation –** during the implementation phase, a plan is developed to put the modification into effect.
5. **Test –** systems testing tests the interfaces between programs to ensure that the system meets all of the original requirements.
6. **Acceptance –** acceptance testing is done on a fully integrated system, and is performed by the customer.

Description Continued:

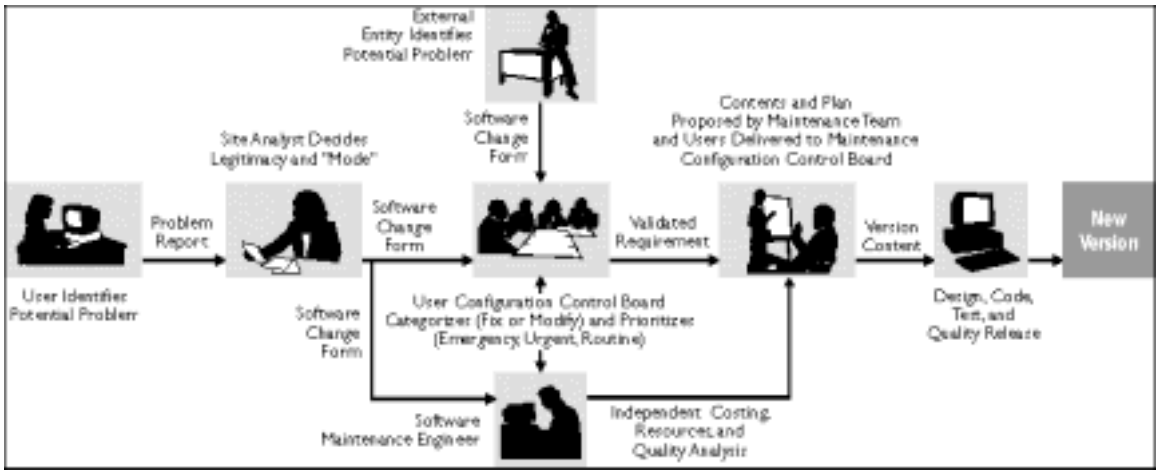
7. **Delivery –** during this phase, the project manager delivers the new system to the customer for installation and operation.

Characteristics	Development and major enhancements projects	SMR (Small Maintenance Requests)
Complexity *(high/low)*	Necessitates a team of software programmers and users.	One or two people can be sufficient
Size *(large /small)*	Necessitates financial , organizational and structural planning.	The effort represents only few hours or few days
Visibility *(important/less important)*	Necessitates corporate visibility.	Limited to operations
Controls *(formal / informal)*	Necessitates a project manager, a control committee.	A first-level supervisor is sufficient.
Approval *(formal / informal)*	Necessitates the approval of senior management.	Operational management approval is sufficient.
Organization *(structured / non structured)*	Necessitates a project management approach.	Necessitates a queue management approach .

Characteristics of projects vs small maintenance requests



The Maintenance Process



Evaluate System:

Both at regular intervals and an ad-hoc basis, benefits and costs are measured and compared with design objectives. Service levels are also measured and evaluated. This evaluation may lead to proposed changes or enhancements.

Assess Changes or Enhancement Requests:

Requests may result from formal evaluation of the system, but more often they come from user, or information technology departments. The impact of each request must be assessed before the steps needed to carry out the changes can be defined.

Analyze the Nature of the Change:

The scope of the requested change must be analyzed. The change may affect the structural aspects of the system (e.g., a change in the system software parameters related to performance, or a new release of the operating system),the design of the system (e.g., a change in the screen layout), or even the business model on which the system is based (e.g., a business change, such as a change in the logic of a process).

Analyze the Impact of the Change:

The impact of any requested change may be extensive and may even include effects on other systems. This task includes identifying the earliest point in the development path that is affected by the change, and then identifying the deliverables that are affected by tracing the task dependencies. With this information, developers can determine which tasks of each stage will have to be repeated and can estimate the cost of doing so. The quality of impact assessment is to a large extent determined by the availability of a repository that stores and relates development information.

Execute the Change:

Select appropriate tasks from the ADM to execute the "change." The impact analysis described above identifies the tasks of planning, analysis, design and construction that must be carried out in order to realize the change. These tasks are now executed in exactly the same way as they were originally carried out. Implementation related tasks often must be carried out as well.

What Kinds of Changes Are Being Made?

To answer this question, we developed the software change taxonomy shown in Table 2. It includes 10 types of changes and root causes for each change type.

Computational

- Incorrect operand in equation
- Incorrect use of parentheses
- Incorrect/inaccurate equation
- Rounding or truncation error

Logic

- Incorrect operand in logical expression
- Logic out of sequence
- Wrong variable being checked
- Missing logic or condition test
- Loop iterated incorrect number of times

Input

- Incorrect format
- Input read from incorrect location
- End-of-file missing or encountered prematurely

Data Handling

- Data file not available
- Data referenced out-of-bounds
- Data initialization
- Variable used as flag or index not set properly
- Data not properly defined/dimensioned
- Subscripting error

Output

- Data written to different location
- Incorrect format
- Incomplete or missing output
- Output garbled or misleading

Interface

- Software/hardware interface
- Software/user interface
- Software/database interface

What Kind of Changes are Being Made

Software/software interface
Operations
COTS/GOTS software change Configuration control
Performance
Time limit exceeded Storage limit exceeded Code or design inefficient Network efficiency
Specification
System/system interface Specification incorrect/inadequate Requirements specification incorrect/inadequate User manual/training inadequate
Improvement
Improve existing function Improve interface

Table 2. *Software change taxonomy.*

We categorized the changes delivered in the last eight releases using this taxonomy. This